Adaptive Data Quality for Persistent Queries in Sensor Networks

Vasanth Rajamani and Christine Julien

Department of Electrical and Computer Engineering
The University of Texas, Austin
{vasanthrajamani, c.julien}@mail.utexas.edu

Abstract. Wireless sensor networks are emerging as a convenient mechanism to constantly monitor the physical world. The volume of information in such networks can be extremely large. And, to be meaningful to applications, this information must be processed at the right level of accuracy. However, there is an inherent trade-off between achieving a high degree of data accuracy and the communication overhead associated with achieving it. We present a simple mechanism for spatially approximate query processing. We present a protocol that leverages gossip based routing to collect network data from a randomly selected set of nodes at a user-defined level of accuracy. We extend this protocol to address persistent queries, long running queries where network data is collected periodically, by treating a persistent query as a temporal aggregate of individual queries. Finally, we provide a novel protocol that dynamically adapts its accuracy based on the quality of the responses to individual requests in the persistent query. We describe this protocol in detail and evaluate its performance through simulation.

Keywords: Adaptive fidelity, gossip.

1 Introduction

Sensor networks have been deployed in a wide range of applications that monitor the physical world in real time such as intelligent construction sites, habitat monitoring, and industrial sensing [19]. When sensor networks are deployed on a large scale, however, there is an explosion in the amount of data to observe and analyze. Requirements on the quality of data collected varies by application and environmental changes. For example, a foreman might evaluate the safety of a construction site by observing the movement of equipment in the site. A periodic summary of data in the site may suffice until a construction truck moves into the danger circle of a crane. At this point, more information should be obtained and the worker should be warned about the danger in his environment. The data fidelity should be increased only when behavior of interest is detected. In this paper, we present a technique that automatically tunes the fidelity of data based on user specifications for such applications.

We address two types of queries that are of value in sensor networks: *one-shot queries* and *persistent queries*. A one-shot query is a one-time occurrence

N. Bartolini et al. (Eds.): QShine/AAA-IDEA 2009, LNICST 22, pp. 131–147, 2009.
© Institute for Computer Science, Social-Informatics and Telecommunications Engineering 2009

in which the application requests data values from some or all of the nodes in the network. This query has no relationship to other queries that may be issued by the application. On the other hand, a persistent query is a long-lived operation that provides periodic responses. We implement persistent querying as a temporal aggregation of component one-shot queries. Doing so allows us to use results of component one-shot queries to influence the behavior of subsequent component queries.

Sensors are typically battery operated and hence extremely resource-constrained. Communication costs account for a large amount of the battery drain during operation, and sending large amounts of unnecessary data reduces the network lifetime substantially. Two approaches have been proposed to reduce the query communication overhead—in-network aggregation (e.g., [14]) and approximate querying (e.g., [17]). In-network aggregation techniques typically build and maintain a tree over the network and distribute the aggregation operation along all non-leaf nodes of the tree. In approximate query processing, the response is typically provided to the user as an estimation of the correct answer with deterministic or probabilistic guarantees quantifying the confidence in the estimate. Both, in-network aggregation and approximate query processing (AQP) have some attractive properties. In-network processing uses actual data collected from sensors. AQP typically models the data at a base station and periodically updates the stored model using values from the network [4]. Other approaches [20] form spatially correlated groups, and one node per group participates in the query resulting in fewer messages being transmitted.

In networks with dynamic data values, it is beneficial to retrieve actual data values and be less reliant on an *a priori* model. In this paper, we present a querying technique that provides approximate querying by selecting a subset of nodes and using actual data from these nodes at query time. This frees our approximation protocol from maintaining state information on the nodes and also alleviates the need for all nodes to participate in every query. While it is possible to accomplish this approximation using any number of underlying networking protocols, we choose gossip routing [3,11]. In its basic form, on receiving a packet, a node chooses to retransmit or drop a packet based on a threshold parameter, p. When a node receives a query, if it chooses to retransmit the query, it actively takes part in resolving the query; otherwise it drops the packet and reduces the likelihood that its downstream neighbors participate. Such a protocol is inherently approximate, as the number of nodes participating varies probabilistically depending on parameter p. In addition to adapting itself nicely to AQP at a conceptual level, gossip routing is robust to changes. Current in-network aggregation and approximate algorithms tend to maintain a tree or a cluster based overlay in which the failure of an intermediate node can lead to significant overhead in rebuilding the aggregation framework. Gossip routing does not impose any hierarchy on the network, and the overhead of performing successive queries is impacted less by the node failures that are fairly common.

Another advantage of using gossip routing is that it allows us to make no assumptions about the data or network characteristics. Some approximation algorithms cluster nodes with highly correlated sensor values [20] which requires *a priori* knowledge of the range of data values in the network. Since we make no assumptions about the data or network characteristics, we instead associate *data quality metrics* with query responses. These metrics may include the number of nodes participating, the variance of sampled data, and the spatial distribution of the sampled nodes. For one-shot queries, the user can use this quality metric as a yard stick by which to analyze his results [15]. For persistent queries, where the query measures sensed conditions over a period of time, we use these data quality metrics to adapt the query's intended fidelity automatically. In this paper, we present a mechanism to perform *adaptive approximate query processing.* Because we view a persistent query as an aggregate of one-shot queries, we use the data quality metrics associated with individual one-shot queries to adapt the fidelity of the protocol for subsequent one-shot queries. Sensor network queries can be broadly classified into two types— aggregate and stream queries. Aggregate queries provide a single aggregated answer, like the average value of the sensor network. Stream queries typically return a stream of data values from different nodes in the network. In this paper, we focus on providing a protocol for adaptive approximate query processing for aggregate queries. Previous work on gossiping [1,5,16] has focused on computing aggregates in a completely distributed fashion. In contrast, we employ gossiping to retrieve a subset of data and control the degree of hosts involved by analyzing the collected data (answers to component queries in the persistent query) in a centralized manner.

The novel contributions of this work are as follows. First, we propose a protocol that incorporates gossip routing for spatially approximate query processing. Second, we discuss the impact of exposing various data quality metrics and underscore how an application can use them to interpret the quality of a response. Third, we show how to incorporate data quality metrics to adapt the accuracy of the AQP algorithm for persistent queries. Finally, we evaluate our protocols and verify their utility.

The rest of this paper is organized as follows. Section 2 adapts gossip routing to AQP. Section 3 evaluates gossip routing for the task of AQP. Section 4 provides a mechanism to expose data quality and uses it to provide context. Using the insights gained from evaluating the AQP protocol, Section 5 provides a protocol that performs adaptive AQP, and Section 6 evaluates its performance. Section 7 discusses related work, and Section 8 concludes.

2 Gossip Routing Based AQP

In this section, we describe how gossip routing provides approximate responses to applications' queries. Gossip routing is based on probabilistic broadcasting, in which a predetermined threshold, p, determines whether a node rebroadcasts or drops a received packet. If p is one, then the behavior is equivalent to flooding. In most networks, setting p to a value smaller than one can still result in a packet reaching all nodes in the network with a very high probability. In gossip routing, only a subset of nodes are involved in query execution. If the query is executed several times, this subset is likely to be different each time, thereby spreading the load more evenly. However, since all nodes in the network do not participate in every query, the result obtained is inherently approximate. In the rest of this section, we explain how we adapt gossip routing to suit our needs.

We first evaluate a basic gossip protocol for providing approximate query results. The state variables for each host in our protocol are shown in Fig. 1. Only the state for a single query is shown; each query has a duplicate set. Our protocol uses two types of packets: *Query* packets and *QueryReply* packets. A *Query* packet looks like:

id	– A's unique host identifier
neighbors	– A's logically connected neighbors
parent	– A's parent in the tree
p	– A's probability threshold for broadcasting an incoming query
data	– A's data value obtained from its sensors

Fig. 1. State Variables for Protocol on Node A

$$\langle query\_id, p, data\_request, sender, originator\rangle.$$

The *query_id* is used to ensure a node does not respond to or forward the same query twice. The p value is the probability with which a receiving node should retransmit the packet. The *data_request* contains the application's data needs (e.g., the type of sensor reading desired). The *sender* of a query is the node that forwarded the packet, while a query's *originator* is the query issuer. A *QueryReply* packet simply contains the data that is the response, the unique query id number, and the id of the destination host (i.e., the query issuer):

$$\langle query\_id, data, destination\rangle.$$

These two packet types are kept necessarily simple to accommodate resource-constrained networks. To define the protocol's behavior, we use I/O Automata notation [13]. We show the behaviors of a single host, A, indicated by the subscript A on each behavior. Each *action* (e.g., $QueryReceived_A(q)$ in Fig. 2) has an effect guarded by a precondition. Actions without preconditions are *input actions* triggered by another host. Each action executes in a single atomic step. We abuse I/O Automata notation slightly by using, for example "send *Query* to *neighbors*" to indicate a sequence of actions that triggers the QueryReceived action on each neighbor.

The basic gossip protocol is very simple. When a node receives a query, it first logs the query's sender ($q.sender$) as its parent and sends its sensor data through its parent to the query issuer. It then uses the probability p to determine whether it will forward this query to its neighbors. To prevent nodes from processing the same query multiple times, a node checks whether it has received the query previously (based on the query's unique id) before processing it. Fig. 2 shows this behavior in I/O Automata form. Also shown in the figure is a node's behavior in response to receiving a *QueryReply*; the node checks if it is the targeted destination; if not, the node forwards the packet to its parent. This is a slightly simplified version of the protocol that only considers a single query from a single application that is active at a given time in the network. The protocol's implementation maintains additional state to sort out the forwarding information associated with different active queries.

```
QueryReceived_A(q)
  Effect:
    if !received(q.query_id) then
      parent := q.sender
      r = ⟨q.query_id, data, q.originator⟩
      send QueryReply(r)
      p := rand()
      if p ≥ q.p then
        q.sender = A
        send Query(q) to neighbors
      end
    end

QueryReplyReceived_A(r)
  Effect:
    if r.destination = A then
      //*** send r.data to application ***//
    else
      send QueryReply(r) to parent
    end
```

Fig. 2. Gossip based AQP

Fig. 3 illustrates the protocol with an example. The value inside a circle is a node's data value. The client is the node in the center—the circle containing a PDA. In the figure, the dashed lines correspond to query replies, while the thick lines indicate a query being forwarded. The tuples next to the response line show the sensor readings carried by the query reply packets. Nodes with concentric circles dropped the packet. On receiving a query from the application, the client broadcasts it to its neighbors $\{c$, f, g and $h\}$. The query contains a probability threshold specified by the application. In this example, nodes f and h decide to drop the query. They each create a reply packet containing their node identifier and data value and send it back to the client. (The query identifier has been omitted from the figure for brevity.) Based on the probability p, nodes c and g choose to forward the query. Both create reply packets they send back to the client. Node c forwards the query to b and d. When node c receives a reply from b or d, it simply forwards it to its parent, the querier.

An application can aggregate the collected values or use individual readings depending on its needs. A biologist checking the number of animals in the habitat might be satisfied with an aggregate sum value. On the other hand, he might want to build a map of their movement by using information from all the data in his query response. Each query might probabilistically choose a different set of sensors and, over a period of time, the biologist can build a complete picture without having to query all the nodes every time. Since imprecision is an inherent part of any approximate algorithm, we expose data quality metrics to provide context to the query response. A very simple data quality metric is the number of nodes that participated in the query. Consider a query where the user is

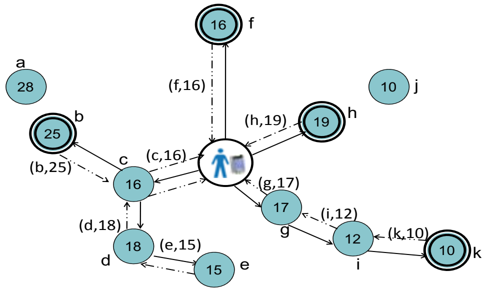

Fig. 3. Protocol example

interested in sensor readings from nodes placed on cranes in an industrial site. If he receives a stream response with just two crane values, when there are 10 cranes visible, he might choose to reissue the query with a higher p. While counting the number of animals, the biologist will feel a lot more secure about the query response he receives if he is provided a confidence interval along with the average humidity. This can be done by exposing the data's variance and the number of nodes sampled. In the rest of this paper, we focus on situations where it is beneficial to aggregate the data values returned from the sensor nodes. This protocol works on the assumption that the client device provides the probability p as an input. In the next few sections we show that changing p does in fact affect the accuracy of the results. We also show how this simple protocol can be leveraged to adaptively tune the accuracy of the result for persistent queries by using the data quality metrics exposed.

3 Effectiveness of Using Gossip Routing for AQP

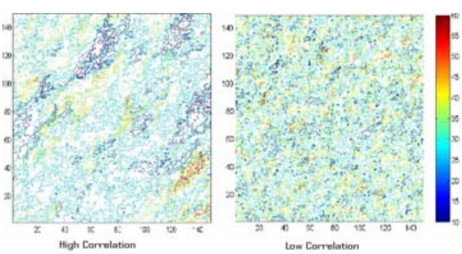

Fig. 4. Correlation of Sensor Grids

Our gossip protocol assumes that, given a good value of p, one can leverage gossip routing to perform approximate query processing. In this section we study the impact of changing p and ascertain whether this assumption holds in different environments. We also develop insights on how to manipulate p to accommodate different application requirements. To thoroughly evaluate our protocol, we used the TOSSIM network simulator [12], which allows direct simulation of TinyOS [6] code written for MICA2 motes.

3.1 Data Set

For modeling sensor data we used a tool provided by Jindal and Psounis [9][1]. The tool generates spatially correlated synthetic data for sensor networks of varying sizes. The data traces generated have been shown to be very close to physical phenomenon observed in the real world. Since our goal is to investigate the feasibility of using gossip routing to perform approximate query processing, this tool provides us a convenient way to test under different simulated environments. We generated spatially correlated sensor data for a 150m x 150m grid. The tool takes in an input parameter β which allows us to manipulate the degree of spatial correlation in the sensor network. A higher value of β makes a node more likely to choose a data value independent of its neighbors', thus producing spatially uncorrelated data. We varied the value of β to 0.001, 0.018 and 0.33, producing sensor networks with high, medium and low data correlations respectively. Fig. 4 shows example surface plots of two data traces we used; the plot on the left shows highly correlated data, while the right shows a data trace that is significantly less correlated.

3.2 Simulation Setup

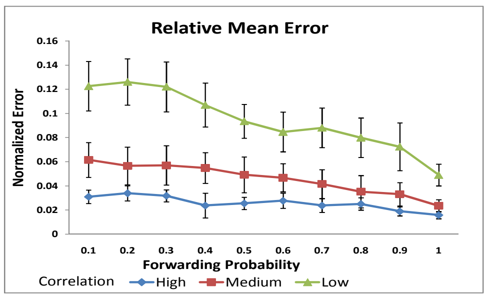

Fig. 5. Accuracy as a function of p

We generated synthetic traces corresponding to the distribution of temperature data in a sensor network field. Given the synthetic data, we explore whether using gossip routing is effective in performing approximate query processing. The protocol is used to query data from the network and compute the value of the average temperature. We used a uniform random placement of sensor nodes. To model the radio transmissions, we used TOSSIM's disc model with a radius of 10 feet. The number of sensor nodes in the network was set to 100. Error bars indicating 95% confidence intervals are included in the graphs whenever possible.

We posit that increasing p will increase the accuracy of our protocol. Fig. 5 confirms that the relative mean error across all responses does decrease as p becomes larger. The accuracy increases because the number of nodes responding with a data value increases as p gets larger. It increases from about 5 to 23 nodes as p varies between 0.1 and 1. The figure plots the normalized error against different values of p for data sets with three different correlation levels. The absolute error is the difference between our protocol's computed average and the actual average provided by an oracle. The normalized error is the absolute error normalized by the correct average provided by an oracle. The normalized error decreases as p increases, regardless of the data distribution, although the

[1] The source code for the tool is provided at `http://www-scf.usc.edu/~apoorvaj/tool.html`

decrease is much more pronounced when the underlying data is less correlated. This is intuitive because, when all nodes have more or less the same data value, sampling more nodes will not produce a large change in the final answer. We can infer from this graph that using a large p is of limited value when the data is highly correlated. Even when querying very few nodes, by setting p to a low value, a gossip protocol can produce an answer very close to the correct response. This is one of the key insights we use in Section 5 to automatically adapt the protocol for persistent queries even when there is no.

However, increasing the number of nodes involved in a query comes with the overhead of increased number of transmitted messages. Fig. 6 plots the increase in overhead associated with raising the value of p. For example, when p is increased from 0.5 to 1, the number of messages transmitted increases almost seven fold. As radio transmission accounts for a large amount of the energy consumption in battery powered devices, choosing the right value of p provides a good lever to trade energy for accuracy while performing approximate query processing.

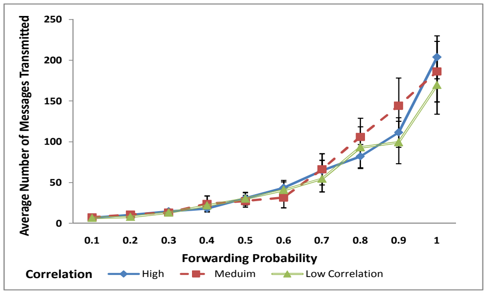

Fig. 6. Overhead vs. p

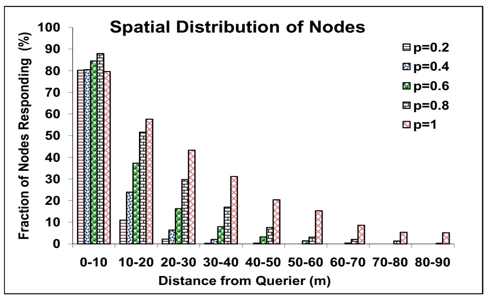

Fig. 7. Spatial Distribution of Participants

In addition to the number of nodes participating, an important factor to consider is the spatial distribution of the nodes that respond. Fig. 7 shows the spatial distribution of the nodes participating in a query. The bins on the horizontal axis represent the distance from of the responding nodes the querier. For example 20-30 represents nodes between 20m and 30m from the query issuer. The vertical axis is the percentage of nodes that responded out of all nodes that were reachable from the querier at that distance range. When the retransmission probability is low, the response obtained is a local one, i.e., it is biased towards nodes close to the querier. However, even when the retransmission probability is high, the percentage of nodes responding to the query decreases as we move away from the querier. This effect is not only due to the value of p but is also impacted by packet collisions. This is confirmed by the fact that the percentage drops to a low value even when the network is being flooded. The farther a node is from the querier (e.g., nodes in the 80-90 bin are about 9

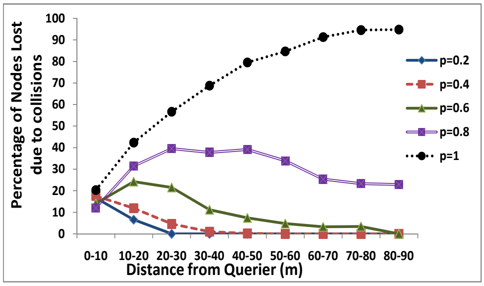

Fig. 8. Difference between Ideal and Observed Spatial Distribution

hops away on average), the greater the chance of a collision related packet drop either during query or response prorogation.

To isolate the impact of p alone on the spatial distribution of the query, we wrote a version of our protocol in Java and compared the results with the results generated using TOSSIM. Fig. 8 plots the difference between what we achieved and what we could have achieved ideally. Each data point represents 100 runs in the simulator. From the figure we can see that collisions have significant impact when the value of p is high. This behavior is acceptable for the *immersive* sensor network applications we target since the nature of the query is often localized around the query issuer. Remote distributed sensing applications, like long term industrial monitoring, require a smooth sampling of nodes by using fine-grained notions of location to route sampling messages. However, this requires the protocol to have prior knowledge of the application scenario, the physical confines of its operating area, and fine-grained location information for every node. While our approach to approximate query processing can be adapted for remote distributed sensing applications by setting a relatively high value of p, it is better suited to a vast number of *immersive* sensor network applications, that can be deployed in an ad hoc fashion to create a picture of more local data.

4 Data Quality Metrics

Imprecision is an intrinsic part of any approximate query processing system. Different quality metrics such as the number of nodes participating in the query, the variance of the underlying data, and the spatial distribution of the nodes provide the application different types and amounts of confidence when interpreting a query response. While this information can be useful for one-shot queries in helping the application determine the usefulness of the returned data [15], it can be even more beneficial to persistent queries that can adapt their querying behavior over time. We expose *data quality metrics* associated with a collective response to a query that can influence the subsequent course of action. Some example data quality metrics for aggregate sensor queries are:

- *Number of Nodes Participating (N)*: Knowing that a large number of nodes participated in the query may be sufficient to represent the quality of a returned query result. If the number of nodes is too low, an application may choose to re-issue the query with a higher p.
- *Variance (V)*: Knowing the variance across the returned data samples can also be very useful to an application. If the variance is low, the application may issue subsequent similar queries with a much lower value of p to reduce overhead. A high variance within data that is expected to be correlated indicates a poor sampling, and subsequent queries will benefit from a larger p.
- *Locality (L)*: When sensor nodes are able to attach location to their readings, exposing the data's spatial distribution can add useful context. An easy alternative is to expose the distribution of the hop counts from the querier. This can give a good intuition for how spatially distributed are the nodes contributing to the response. A user might be able to fine tune his results successfully by just varying p. Alternatively, he may decide that the best way to get non-local results is to flood the network or use a sophisticated adaptation function (discussed in detail in Section 5), if he is interested in long term sensing style results.

These quality metrics can be exposed with very little additional computation or communication overhead. In the next section, we focus on using these metrics to dynamically tune the sequence of queries that constitutes a persistent query.

5 Adaptive Approximate Querying Protocol for Persistent Queries

We implement a persistent query as a sequence of one-shot queries. We refer to a particular one-shot query within a persistent query as a *round*. In our adaptive model, the protocol can use the data quality metrics associated with the previous rounds to parameterize the protocol's execution for the next round. Using the data quality metrics exposed, an application developer can write an *adaptation function* that dynamically changes the behavior of a protocol for persistent queries, after considering data dynamics and user preferences. In this section we demonstrate the feasibility of our approach using an example adaptation function.

5.1 Adaptation Function

One can write complex functions in which combinations of locality and variance influence adaptation. However it becomes difficult for the application to express domain knowledge in a straightforward manner. Often, a simple function can capture the essence of the required adaptation. For example, an application that monitors chemical leaks must decide if the data obtained in any given round is significantly different from the previous rounds and change the query behavior accordingly. A good adaptation function should specify the value beyond which a chemical leak becomes dangerous, and tune the next round of the query based on the response obtained. We present one such adaptation function; it is conceptually simple and yet shows a high degree of success during adaptation. In this section, we focus on queries that obtain the approximate average value of the network, as it is the most typical summary statistic used in long term monitoring applications. Our adaptation function uses the confidence intervals of the average to change the retransmission probability in the next round if necessary. Confidence intervals, are often used to signify the likely range of an estimated value based on some samples from the data. The standard formula for computing the confidence length:

$$\mathit{ConfidenceLength}(\mathit{CL}) = 1.96 * \sigma/\sqrt(n)$$

σ is the standard deviation of the data, and n is the number of samples. The constant 1.96 indicates 95% confidence in the computed estimate.

Confidence lengths can be calculated easily but are an unintuitive way to express user preferences. It is easier for an application to express the extent of error it is willing to tolerate. We call this the *Tolerable Error*:

$$\mathit{TolerableError}(\mathit{TE}) = 100 * \mathit{CL}/\mu$$

TE can be easily expressed by the application as a single value. For example, a value of 10% indicates that the confidence length computed from a query response should be no more than 10% of the mean of the samples. A confidence length is small only when a large number of nodes participate or when the data is highly correlated (i.e., the standard deviation is small). Consequently, the error for a query round will be small for the same reasons. We now show how to use a simple adaptation function that employs this *Tolerable Error* to perform adaptive approximate query processing for persistent queries. Fig. 9 updates our query processing algorithm to of expanded state. First, the node stores the application-specified *Tolerable Error* (*TE*) for each persistent query. The state variable p becomes a list of p values, one for each round of the persistent query. They are indexed by i, the number of the round with which the particular p is associated. We also introduce a timer, *queryTimer*, which fires when it is time to issue a new round of the persistent query. It is at this time that the results for the previous round are delivered to the application. Finally, the variable *replyList* stores the samples constituting a round until the round is complete. The protocol shown in Fig. 9 is a simplified version of the actual implementation. We check the query-id before processing a node's reply packet to ensure that all responses belong to the same round.

The figure shows the replacement behavior for the QueryReplyReceived action. Instead of immediately forwarding replies to the application, our protocol stores them in the *replyList*, and waits to aggregate the replies for the application before the next query round. We add the action SendPersistentQueryRound to the formalization. This action is timer driven; when the timer expires indicating it is time to send the next one-shot query for this persistent query, this behavior is enabled. It first computes the average and the error for the samples received in the previous round and sends the result to the application. It then compares the error to the application-specified tolerable error *TE*.

```
QUERYREPLYRECEIVED(r)
  if r.destination = A then
      replyList := replyList ∪ r
  else
      send QueryReply(r) to parent
  end

SENDPERSISTENTQUERYROUND()
  Precondition:
      queryTimer expires
  Effect:
      average := computeAverage(replyList)
      error := computeError(replyList)
      //*** send average and error to application ***//
      diff := TE − error
      if |diff| < 1 then
          increment := 0.05
      else
          increment := 0.20
      end
      if diff > 0 then
          increment := −increment
      end
      p_{i+1} := p_i + increment
      if p_{i+1} > 1 then
          p_{i+1}=1
      end
      if p_{i+1} < 0.1 then
          p_{i+1}=0.1
      end
      reset queryTimer
```

Fig. 9. Updated Query Processing Algorithm

Our example protocol uses the *TE* very simply. If the error is close the tolerable error, the protocol makes only a small adjustment in the value of p (an adjustment with a magnitude of 0.05). Otherwise the protocol makes a bigger step (an adjustment with a magnitude of 0.20). A more sophisticated adaptation would use a continuous adjustment scale, where the magnitude of the increment is computed relative to the magnitude of the *TE* directly. The increment is adjusted based on whether p should be raised or lowered, and then p_{i+1} is calculated. Finally, the value of p is adjusted if it went outside the range of $0 - 1$. This is a simplified example that matches what was used in our experiments. Other adaptation algorithms can be designed that use a larger history (more than just the error in the last round) so that the changes are not as abrupt. Our goal was to demonstrate the efficacy of the technique even when using a relatively simple adaptation function. In the next section, we show that even this simple adaptation protocol is quite capable of dynamically trading message overhead for desired accuracy.

6 Evaluation

In this section, we evaluate the performance of the simple adaptation mechanism outlined in Section 5. This provides an example of how a combination of a parameterizable protocol and data quality metrics can generate a protocol that incurs the least amount of overhead possible while still satisfying application-defined requirements. We assume the application is sampling a field of sensors all measuring the same thing (e.g., the temperature of animals in a farm). The application expects the values to be similar; therefore the deviation of the results from a mean is a reasonable adaptation point. The application provides a Tolerable Error (as described in the previous section), and the protocol adapts p to dynamically target this Tolerable Error. We use the same experimental setup as outlined in Section 3. Once again, we have three types of data sets representing data with correlation varying from high to low. We provide 95% confidence intervals for our results. A persistent query is run for 300 seconds, and a new query round is issued every 25 seconds.

Fig. 10 shows the number of nodes responding to the query as the required Tolerable Error is increased. A low value of tolerable error indicates an application that requires a high degree of accuracy. Conversely, a high Tolerable Error indicates that the application does not require high fidelity data. When the Tolerable Error is very low (left of the graph), a large number of nodes need to be involved to satisfy this requirement. When the requirement is less restrictive, receiving results from far fewer nodes will suffice. In our experiments, p is set to 0.5 during the first round. As the rounds progress, the adaptation function enforces a change in p based on the computed error. If the error is low, p is progressively increased; if it is high, p is decreased. The average value of p for the persistent query varies from 0.92 (1% TE) to about 0.16 (20% TE). Fig. 10 clearly shows that our protocol changes the number of nodes involved (and hence the communication overhead) progressively while taking into account application constraints regardless of the nature of the underlying data. However, it also shows that the

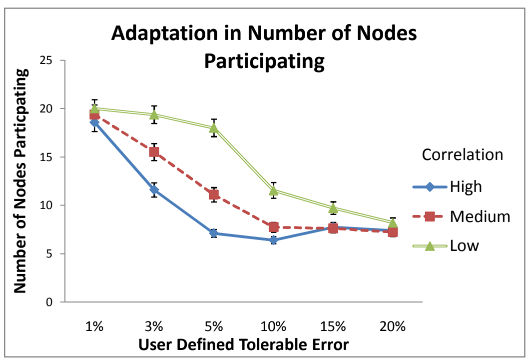

Fig. 10. Adaptation changing the number of nodes in a Persistent Protocol

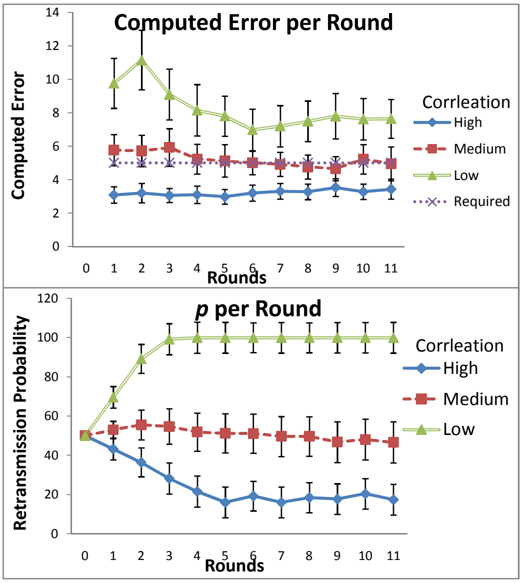

Fig. 11. Adaptation per Round

underlying data can impact the number of nodes required to successfully match user expectations. In most cases, the number of nodes required to get the same result quality when the data is loosely correlated is much more than when the data is highly. When the data is loosely correlated, the standard deviation is high, resulting in a large computed error value.

Fig. 11 takes a deeper look at the adaptation process when the application specifies that the Tolerable Error is 5%. The X axis shows the round number of the persistent query. The Y axis in the top figure shows the computed error at each round. The bottom figure shows the retransmission probability (expressed as a percentage) used by the query for each round. The first round's one-shot query is issued with a probability of 50%. When the data is not correlated, the application is unable to meet the required error of 5% (dashed line in the top graph), and consequently ends up increasing its retransmission probability to 100% very quickly . Even when p is set to 1, the protocol is still unable to attain the user required error, but it does manage to reduce the error from 11% to 8%. The same algorithm behaves differently when the data is moderately correlated. It is relatively close to the desired Tolerable Error at about 50% retransmission probability. Consequently, it reduces its retransmission probability in small amounts until it reaches the desired degree of data quality. Once it satisfies the application's accuracy requirements, it hovers around that value for the remaining rounds. Given highly correlated data, our adaptive protocol can achieve the application's Tolerable Error easily with the starting value of p set to 50%. Consequently, it tries to minimize the number of nodes involved in query processing. As can be seen from the top graph, the computed error increases slightly as the rounds progress but remains well below the Tolerable Error. The bottom graph shows that the retransmission probability drops drastically from 50% to about 17% by the end of 12 rounds. This translates to only about six nodes being involved in the query. Thus, the protocol has adaptively traded a slight loss in accuracy for a large savings in communication overhead because the accuracy loss was well within application tolerable levels.

The data used in our experiments is not jointly Gaussian. In spite of that, confidence intervals have proven to be a good point of adaptation. This suggests that gossip routing can be successfully parametrized for a large number of data distributions. From these results it can be seen that there is a large benefit to be gained from dynamically adapting the behavior of a persistent protocol based on results gathered in the previous rounds. Allowing the application to specify accuracy constraints and using that as a benchmark to adapt a protocol's behavior dynamically can lead to an ideal number of nodes answering a query. This helps answer the query effectively within the bounds of application tolerability and reduce communication overhead.

7 Related Work

Our work is broadly related to three classes of systems that exist in the literature.

Approximate Query Processing: Since performing in-network aggregation [14] by distributing the computation through out the network can be quite expensive, approximate querying techniques were designed to provide estimates of answers. CAG [20] creates clusters where nodes with highly correlated data form a group, and only the cluster head is involved in transmitting data. CAG's emphasis is network structure maintenance while ours is to adapt the approximation technique based on the dynamics of the network. Also, we avoid the overhead associated with maintaining grouping mechanisms like trees or clusters. Other approximate query processing algorithms [4,7] create models of data at the base station, and the querier interacts with the base station. The base station maintains estimates of the data at the sensor nodes and employs different techniques to keep its estimate accurate with the actual data. Our approach queries actual data at query time and also does not require any state maintenance mechanism to compute the estimates. In addition, we expose data quality metrics to add context to a query response. Finally, Backasting [18] is a technique where adaptive sampling is used to perform estimation of a spatial field and identify interesting objects, e.g., the boundary of a physical space. Adaptation is used to determine if a region is of interest by sampling a few nodes initially and then imposing a hierarchy. We focus on using adaptation over an extended period of time for persistent queries and impose no hierarchy.

Gossip Routing: We chose a gossip routing based protocol because it naturally lends itself to selectively sampling data from nodes. There has been extensive research in using the concept of gossiping for a variety of tasks. Several researchers have incorporated gossip routing in the sensor networks domain [3,5]. Their focus is typically on studying the coverage of gossip routing for different network topologies. In contrast, we focus on using gossip routing as a mechanism to perform adaptive approximate query processing. One interesting variant is where nodes update the probability of retransmission based on the relationship between nodes in the network hierarchy [11]. Nodes are inferred to be organized as parents, children or siblings and these relationships are used to tune the retransmission probabilities. This is complementary to our work and can be used in conjunction to adapt our protocol to network topology and data distribution simultaneously. Gossip routing has also been used to perform distributed aggregate computation [10] by making nodes gossip which leads to an eventual convergence on a common value. The convergence rates of these algorithms is typically pretty slow.

Query Consistency: There has been some recent work in assessing the validity of a query response highlighting how network disruptions can render the answer to a query completely arbitrary [2,8,15]. Most of these systems use validity metrics to give an idea of the correctness of the response in the presence of node failure. Our data quality metrics provide relatively cheap context along with a query response *and* use this context directly for automatic adaptation in persistent queries.

8 Conclusion and Future Work

We presented a simple yet effective protocol to perform approximate query processing by leveraging gossip routing. We exposed meta-data in the form of quality metrics and demonstrated how they add context to a query response in both one-shot and persistent queries. Finally, we provided a protocol that uses the data quality metrics to automatically adapt approximate query processing for persistent queries. Our results demonstrate that we can effectively trade off user defined accuracy for overhead. In future, we plan to run our protocol on a real sensor network deployments and plan to incorporate temporal approximation. In addition we plan to incorporate the benefits of geographic scoping as shown in [5,16] into our adaptive algorithm.

References

1. Bash, B., Byers, J., Considine, J.: Approximately uniform random sampling in sensor networks. In: Proc. of Wkshp on DMSN, pp. 32–39 (2004)
2. Bawa, M., Gionis, A., Garcia-Molina, H., Motwani, R.: The price of validity in dynamic networks. Proc. of ACM SIGMOD 73(3), 245–264 (2007)
3. Braginsky, D., Estrin, D.: Rumor routing algorthim for sensor networks. In: Proc. of WSNA, pp. 22–31 (2002)
4. Deshpande, A., Guestrin, C., Madden, S.R., Hellerstein, J.M., Hong, W.: Model-driven data acquisition in sensor networks. In: Proc. of VLDB, pp. 588–599 (2004)

5. Dimakis, A.G., Sarwate, A.D., Wainwright, M.J.: Geographic gossip: efficient aggregation for sensor networks. In: Proc. of IPSN, pp. 69–76 (2006)
6. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D.E., Pister, K.S.J.: System architecture directions for networked sensors. In: Proc. of ASPLOS, pp. 93–104 (2000)
7. Jain, A., Chang, E.Y.: Adaptive sampling for sensor networks. In: Proc. of DMSN, pp. 10–16 (2004)
8. Jain, N., Kit, D., Yalagandula, D.M.P., Dahlin, M., Zhang, Y.: Network imprecision: A new consistency metric for scalable monitoring. In: Proc. of OSDI (2008)
9. Jindal, A., Psounis, K.: Modeling spatially correlated data in sensor networks. ACM TOSN 2(4), 466–499 (2006)
10. Kempe, D., Dobra, A., Gehrke, J.: Gossip-based computation of aggregate information. In: Proc. of IEEE FOCS, pp. 482–491 (2003)
11. Kyasanur, P., Choudhury, R.R., Gupta, I.: Smart gossip: An adaptive gossip-based broadcasting service for sensor networks. In: Proc. of MASS, pp. 91–100 (2006)
12. Levis, P., Lee, N., Welsh, M., Culler, D.: TOSSIM: accurate and scalable simulation tinyos applications. In: Proc. of IEEE SenSys, pp. 126–137 (2003)
13. Lynch, N., Tuttle, M.: An introduction to I/O automata. In: CWI-Quarterly, pp. 219–246 (1989)
14. Madden, S., Franklin, M., Hellerstein, J., Hong, W.: TAG: A tiny aggregation service for ad hoc sensor networks. In: Proc. of OSDI, pp. 131–146 (2002)
15. Payton, J., Julien, C., Roman, G.-C.: Automatic consistency assessment for query results in dynamic environments. In: Proc. of ESEC/FSE, pp. 245–254 (2007)
16. Sarkar, R., Zhu, X., Gao, J.: Hierarchical spatial gossip for multi-resolution representations in sensor networks. In: Proc. of IPSN, pp. 420–429 (2007)
17. Skordylis, A., Trigoni, N., Guitton, A.: A study of approximate data management techniques for sensor networks. In: Proc. of WISE, June 2006, pp. 1–12 (2006)
18. Willett, R., Martin, A., Nowak, R.: Backcasting: adaptive sampling for sensor networks. In: Proc. of IPSN, New York, NY, USA, pp. 124–133 (2004)
19. Xu, N.: A survey of sensor network applications. Technical report, The University of Southern California (2002)
20. Yoon, S., Shahabi, C.: The clustered aggregation (CAG) technique leveraging spatial and temporal correlations in wireless sensor networks. ACM TOSN 3(1), 3 (2007)