Snapshot Queries: Towards Data-Centric Sensor Networks

Yannis Kotidis
AT&T Labs-Research
kotidis@research.att.com

Abstract

In this paper we introduce the idea of snapshot queries *for energy efficient data acquisition in sensor networks. Network nodes generate models of their surrounding environment that are used for electing, using a localized algorithm, a small set of representative nodes in the network. These representative nodes constitute a* network snapshot *and can be used to provide quick approximate answers to user queries while reducing substantially the energy consumption in the network. We present a detailed experimental study of our framework and algorithms, varying multiple parameters like the available memory of the sensor nodes, their transmission range, the network message loss etc. Depending on the configuration, snapshot queries provide a reduction of up to 90% in the number of nodes that need to participate in a user query.*

1. Introduction

Sensor networks are used in a wide variety of monitoring applications, ranging from habitat/environmental monitoring to military surveillance and reconnaissance tasks. Although today's sensor nodes have relatively small processing and storage capabilities, driven by the economy of scale, it is already observed that both are increasing at a rate similar to Moore's law.[1] At the same time, it is becoming clear that intelligent management of network's dynamics in sensor networks is fundamental. This is because for many applications, like sensors in a disaster area, nodes operate *unattended*. Such systems must adapt to a wide variety of challenges imposed by the uncontrolled environment in which they operate. One of the major technical challenges arises from the severe energy constraints met in these networks. Nodes are often thrown into hostile environments. In such cases, nodes are powered by small batteries and replacing them is not an option.

As nodes become cheaper to manufacture and operate, one way of addressing these challenges is redundancy. In [2] the authors suggest throwing-in extra redundant nodes to ensure network coverage in regions with non-uniform communication density due to environmental dynamics. In this paper we propose a more fundamental approach for dealing with network dynamics in data driven applications. Like [7] we take the position that nodes should be aware of their environment, including network dynamics. Instead of designing database applications that need to hassle with low-level details, we promote the use of data-centric networks that allow transparent access to the collected measurements in a unified way. When queried nodes fail, the network should self-configure to use redundant stand-by nodes. However, unlike [7] that assumes that any node in the vicinity can replace the failed node, we promote a data-driven approach in which a node can "represent" (replace) another node in a query when their collected measurements are similar, where similarity is expressed in some quantitative way.

As an example, we consider the deployment of nodes over a large terrain for the purpose of collecting and analyzing weather data like temperature, humidity etc. This is a scenario where we expect a lot of correlations among the collected measurements, especially among neighboring nodes. However, in reality, it is difficult to predict what are the real points (locations) of interest before actually deploying sensor nodes over the terrain. In a localized mode of operation, nodes can coordinate with their neighbors and elect a small set of *representative nodes* among themselves. This can be achieved using, for instance, a threshold value T and have the nodes elect a local representative whose value (measurement) is, on expectation, within T of their own. Such a scenario has many advantages for data processing and analysis.

- The location and values of the representative nodes provide a picture of the value distribution in the network. One can extend this technique and use multiple threshold values. Each set of representatives, compiled for a value of T, is essentially a "snapshot" of the network at a different "resolution", depending on T. What is important is that, as we will show, these computations can be performed in the network with only a small number (up to

[1] http://nesl.ee.ucla.edu/courses/ee202a/2002f/lectures/L07.ppt

six) of exchanged messages among the nodes.

- The network snapshot can be used for answering user queries in a more energy-efficient way. We call such queries *snapshot queries*. Snapshot queries include both aggregate queries as well as drill-through type of queries where a small collection of nodes are requested to report their individual measurements.
- An elected representative node can take over for another node in the vicinity that may have failed or is temporarily out of reach. Because selection of representatives is quantitative (based on the value of T and the selected error metric as will be explained) this allows for a more accurate computation than when representatives are selected based only on proximity.
- A localized computation of representative nodes can react quickly to changes in the network. For instance, nodes (including the representatives) may fail at random. It is important that the network can self-heal in the case of node-failures or some other catastrophic events. In a data-driven mode of operation, we are also interested in changes in the behavior of a node. In such case the network should re-configure and revise the selected representatives, when necessary.

In our experiments we compare a network setup that answers snapshot queries against a straightforward implementation that does not use representative nodes. We observe that even-though some background messaging and processing is required for the maintenance of the representative nodes, the lifetime of the network is substantially extended (see Figure 10) because most network nodes can stay idle during a snapshot query. Depending on the query, he have observed reduction of up to 90% in the number of nodes that need to participate in a snapshot query, versus the number of nodes that would be required otherwise.

Our contributions are summarized as follows

- We introduce a data-driven framework for dealing with network dynamics in uncontrolled volatile environments. We show how sensor nodes can maintain, using a localized algorithm, a dynamic view of their environment including other nodes in their vicinity.
- Nodes build compact models that capture the correlations among their measurements and those of their neighbors. These models are simple enough to be implemented on low-end nodes and, as our experiments demonstrate, work well for the applications of interest. For managing the limited buffer space (in the range of a few hundred bytes to a few Kbytes) alloted on a node for these models, we introduce a model-aware cache manager that discovers and augments those models that provide the higher degree of accuracy.
- We introduce a new class of *snapshot queries*. These are user queries that are willing to tolerate a small amount of error, at the benefit of requiring far fewer network nodes for their execution.
- We investigate the effects of various network properties like the node's transmission range, the network message loss as well as parameters like the memory size of the sensors, the correlation among measurements of different sensors and the desirable error threshold. Our experiments demonstrate that our techniques are very robust and consistently provide substantial savings during snapshot queries.

The rest of the paper is organized as follows. In Section 2 we describe related work. Section 3 discusses some of the properties of sensor networks and presents an overview of our framework. In Section 4 we describe the data models built by a sensor for capturing its environment and we present a model-aware cache manager for maintaining these models. Section 5 describes a localized algorithm for electing and maintaining the set of representative nodes to be used for snapshot queries. Section 6 contains the experiments. Concluding remarks are given in Section 7.

2. Related Work

Directed diffusion, presented in [7], is a communication model for describing localized algorithms in the context of sensor networks. In the same paper, the authors present a network configuration, using *adaptive fidelity* algorithms, in which localized algorithms selectively turn off some sensors to conserve system resources, when for instance high rate of collisions is detected in the channel. As individual sensors die, others in the vicinity take their place. Our framework was motivated by these ideas but it differs in that it takes a data-centric approach versus the network-centric viewpoint of [7]. We exemplify the view of sensor nodes as data acquisitional devices [12] and promote the use of data-centric localized algorithms for self-configuring within a volatile computing environment.

In database literature there is an on-going effort for rethinking data processing and optimization techniques in the context of sensor networks. For instance, in-network data aggregation [5, 11, 16, 17] uses a routing tree to partially aggregate measurements on their way to their destination and can reduce substantially the amount of data transmitted by the nodes. The work of [5] allows approximate evaluation of user aggregate queries in sensor networks while minimizing the number of messages exchanged among the sensor nodes. Our techniques are more generic as they capture both aggregate and drill-through queries and are orthogonal to these optimizations. A related line of research deals with

Symbol	Definition
$x_i(t)$	Measurement of Node N_i at time t
$\hat{x_i}$	Estimation of x_i by a node $N_j, j \neq i$
T	Threshold used in computation of representatives
N	Total number of nodes in network
n_1	Number of nodes (representatives) in snapshot

Table 1. Symbols and their definition

decentralized algorithms for aggregate computations with applications in P2P and sensor networks [1, 9].

Our framework also relates to the work in [10, 13, 14] that study the tradeoff between precision and performance when querying replicated, cached data but the underlying foundation is different. The network snapshot we maintain is a result of both data dynamics (e.g. changes in data distributions) as well as network dynamics (node failures, changes in connectivity among nodes due to mobility, environmental conditions etc).

In [6] a model-driven data acquisition framework is proposed that uses a trained statistical model in order to limit the number of sensor readings required to answer a posed query with high confidence. While [6] uses a “global” model to capture dependencies assuming a relatively stable network topology, our framework aims at capturing localized correlations in highly dynamic networks and should scale better in networks consisting of hundreds or thousands of nodes. Another difference is that we are not trying to fit the data into a model but, similarly to [4], try to exploit spatio-temporal correlations among readings from neighboring nodes.

In [3] the authors propose the use of counting sketches to cope with link failures during aggregate queries in sensor databases. In principle, one can use sketches (different from those in [3]) to maintain an approximate model of the nodes. An important difference is that our snapshot is maintained overtime, thus each measurement is a time-series. Our models capture the time dimension by encoding the correlations among the nodes. This is different that approximating the data values using e.g. sketches. As our experiments demonstrate, we can capture trends with just a few samples, while sketches would require continuous re-broadcasting of values for updates, thus defeating the purpose of reducing resource consumption in the network.

3. Overview

Table 1 describes the symbols we are using in our notation. For measurements, we often drop index t and use x_i to refer to the current value reported by node N_i. In order to simplify the presentation, we assume that there is a single measurement x_i collected on every sensor node. In practice there can be as many measurements as the number of sensing elements installed on a node. Our framework will still apply in such cases. The only necessary modification is the addition of a $measurement\_id$ during model computation (see Section 4).

A sensor node N_i maintains a data model for capturing the distribution of values of the measurements of its neighbors. This is achieved by snooping (with a small probability) values broadcast by its neighbor node N_j in response to a query, or, as will be explained, by using periodic announcements sent by N_j. [2] One may devise different mathematical models, with varying degrees of complexity, for this process. The modeling that is described later in this paper has the following desirable characteristics, in the context of sensor networks.

- It is simple both in terms of space and time complexity. Our algorithms can operate when the available memory for storing these models in the sensor is as small as a few bytes.
- We do not make any particular assumption on the distribution of the data values collected by the sensors. The only assumption made is that values of neighboring nodes are to some extent correlated. This is true for many applications of interest like collection of meteorological data, acoustic data etc. Furthermore, by modeling these correlations, we are able to capture trends (like periodicity), with very few samples.
- The derived models can be optimized for different error metrics like the sum-squared error (sse), relative error, the maximum error of the approximation etc.

Using the data model it maintains, sensor node N_i provides an estimate $\hat{x}_j$ of the measurement of its neighbor N_j. Given an error metric $d()$ and a threshold value T, node N_i can *represent* node N_j if $d(x_j, \hat{x}_j) \leq T$. Function $d()$ is provided by the application. Some common choices include (i) relative error: $d(x_j, \hat{x}_j) = \frac{|x_j - \hat{x}_j|}{\max(s, |x_j|)}$, where $s > 0$ is a sanity bound for the case x_j=0, (ii) absolute error: $d(x_j, \hat{x}_j) = |x_j - \hat{x}_j|$ and (iii) sum-squared error: $d(x_j, \hat{x}_j) = (x_j - \hat{x}_j)^2$.

Through a localized election process the nodes in the network pick a set of representative nodes of size n_1. Depending on the threshold value T, the error metric and the actual measurements on the sensors, n_1 can be significantly smaller that N. An example of this process is demonstrated in Figure 1 where the representatives for a simulated network of 100 nodes are shown. Dark nodes in the Figure are representative nodes. There are lines drawn from a node N_i to a node N_j that N_i represents. Nodes with no lines attached to them represent themselves (the default choice).

[2] We use the term “neighboring nodes” in a loose, dynamic sense, to mean any node that can directly transmit to the node of interest. This relationship is, in general, not symmetric.

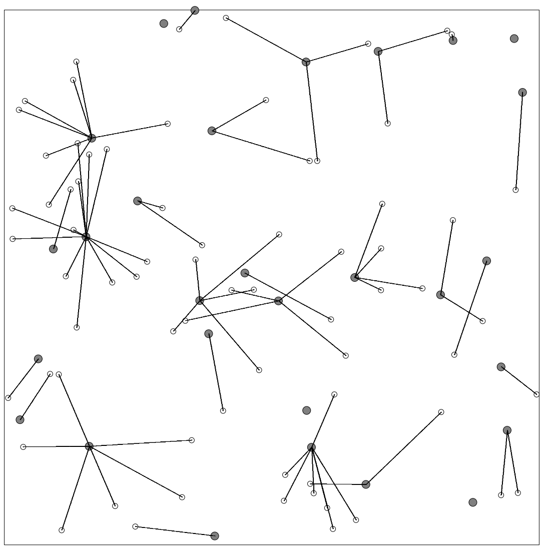

Figure 1. Example of Network Snapshot

An aggregate computation like SUM can be handled by the representative nodes that will in-turn provide estimates on the nodes N_j they represent using their models. Another scenario is to use the representative of a node on an aggregate or direct query, when that node is out-of-reach because of some unexpected technical problem or due to severe energy constraints. Thus, query processing can take advantage of the unambiguous data access provided by the network. Of course, one can ignore the layer of representatives and access the sensors directly, at the penalty of (i) draining more energy, since a lot more nodes will have be to accessed for the same query and (ii) having to handle *within the application* node failures, redundancy etc.

The selection of representatives is not static but is being revised overtime in an adaptive fashion. An obvious cause is the failure of a representative node. In other cases, the model built by N_i to describe x_j might get outdated or fail, due to some unexpected change in the data distribution. In either case, the network will self-heal using the following simple protocol. Node N_j periodically sends a heart-beat message to its representative N_i including its current measurement. If N_i does not respond, or its estimate $\hat{x}_j$ for x_j is not accurate ($d(x_j, \hat{x}_j) > T$) then N_j initiates a local re-evaluation process inviting candidate representatives from its neighborhood, leading to the selection of a new representative node (that may be itself). This heart-beat message is also used by N_i to fine-tune its model of N_j.

Under an unreliable communication protocol it is possible that this process may lead to *spurious* representatives. For instance node N_i may never hear the messages sent by node N_j due to an obstacle in their direct path. It may thus assume that it still represents node N_j while the network has elected another representative. This can be detected and corrected by having time-stamps describing the time that a node N_i was elected as the representative of N_j and using the latest representative based on these time-stamps. In TinyOS nodes have an external clock that is used for synchronization with their neighbors [11]. In lack of properly synchronized clocks among the sensor nodes, one can use a global counter like the epoch-id of a continuous query. This filtering and self-correction is performed by the network, transparently from the application.

3.1. Snapshot Queries

Recent proposals [11, 17] have suggested the use of SQL for data acquisition and processing. The obvious advantage of using a declarative language is greater flexibility over hand-coded programs that are pre-installed at the sensor nodes [12]. In addition embedded database systems like TinyDB can provide energy-based query optimization because of their tight integration with the node's operations.

Basic queries in sensor networks consist of a SELECT-FROM-WHERE clause (supporting joins and aggregation). For instance, in our running example of collecting weather data a typical query may be

```
SELECT loc, temperature
FROM sensors
WHERE loc in SHOUTH_EAST_QUANDRANT
SAMPLE INTERVAL 1sec for 5min
USE SNAPSHOT
```

This is an example of a drill-through query, sampling temperature readings every 1 second and lasting 5 minutes. For this example we assume that each node has a unique location-id *loc* and that nodes are location-aware, being able to process the spatial filter "in SHOUTH_EAST_QUANDRANT". Location can be obtained using inexpensive GPS receivers embedded in the nodes, or by proper coordination among the nodes, using techniques like those proposed in [15]. Location is a significant attribute of a node in an unattended system. For many applications like habitat monitoring, spatial filters may be the most common predicate.

The new extension presented in the query above is the USE SNAPSHOT clause, denoting that the query may be answered by the representative set of nodes. Notice that a node is aware of the location of the nodes it represents, as this information can be passed during the message-exchange round of the protocol we present in Section 5. In this schema, a node provides measurements for the above snapshot query when (i) it is not represented and satisfies the spatial predicate of the query or (ii) it represents another node N_j satisfying the spatial predicate.

The advantage of providing the USE SNAPSHOT directive is that often a much smaller number of nodes will be involved in the processing of the query. A node that is being represented may still be asked to route messages for the query. The probability of this happening can be reduced by having the routing protocol favor paths through representative nodes. Notice that this is feasible, since representatives are physically co-located with the nodes they represent.

An interesting extension is to have each snapshot query define its own error threshold. A straightforward way to handle this case is to start an election process (see Section 5) for each such query. Since each election cycle requires up to six messages per node (and usually a lot less), this is a reasonable startup cost considering the savings for a long-running (continuous) query when executed through the snapshot. The data models, described in Section 4, will be shared among all running queries. Of course some optimizations are possible in a multi query scenario. For instance given queries $Q_1, Q_2, \ldots$ with error thresholds $T_1 \leq T_2 \leq \ldots$ we can obtain a single set of representatives (snapshot) for the most tight threshold T_1 and use them for answering all other queries. We defer this discussion to the full version of this paper.

4. Model Management

Each sensor N_i maintains a small cache containing past measurements from its neighbors. These values are used for building simple models of the correlations between their measurements. We assume a limited amount of memory available for maintaining this cache and any necessary values needed for bookkeeping (e.g. model parameters). Since the purpose of the cache is to provide input for the models built, we are employing a model-aware cache management policy that aims at increasing the accuracy of these models. We will first describe the modeling used in more detail and then present the cache admission and replacement algorithms.

Modeling Correlations. In order to be as general as possible, we do not assume a particular distribution for the data measurements but instead focus on the correlations among neighboring nodes. Such correlations are the motivation of using redundant nodes at the first place [2, 7].

Given $n > 1$ pairs of values for measurements $x_i(t)$ and $x_j(t)$: $(x_i(t_1), x_j(t_1))$,... $(x_i(t_n), x_j(t_n))$ we model the values of $x_j(t)$ as a linear projection of the values of $x_i(t)$. Thus, $\hat{x}_j(t) = a_{i,j} \times x_i(t) + b_{i,j}$. The computation of parameters $a_{i,j}$, $b_{i,j}$ depends on the error metric used. A common choice is the sum-squared error (sse) of the approximation $\sum_{1\leq k \leq n}(\hat{x}_j(t_k) - x_j(t_k))^2$. In that case, the optimal values for these parameters are provided by the following Lemma.

Lemma 1 *The optimal values for $a_{i,j}$, $b_{i,j}$ that minimize the sum squared error of the approximation $\hat{x}_j(t) = a_{i,j} \times x_i(t) + b_{i,j}$ (for t in $\{t_1, \ldots, t_n\}$) are*

$$b^*_{i,j} = \frac{\sum_{1\leq k\leq n} x_j(t_k) - a^*_{i,j} \times \sum_{1\leq k\leq n} x_i(t_k)}{n}$$

and

$$a^*_{i,j} = \frac{n\sum_{1\leq k\leq n} x_i(t_k)x_j(t_k) - \sum_{1\leq k\leq n} x_i(t_k)\sum_{1\leq k\leq n} x_j(t_k)}{n\sum_{1\leq k\leq n} x_i^2(t_k) - (\sum_{1\leq k\leq n} x_i(t_k))^2}$$

Proof: We consider a set of n points $\mathcal{P}$ in the plane. In particular $\mathcal{P} = \{(x_i(t_k), x_j(t_k)), k = 1, \ldots, n\}$. Let $y = a \times x + b$ be a line in the plane. The sum-squared error $\sum_{1\leq k\leq n}(\hat{x}_j(t_k) - x_j(t_k))^2$ of the approximation $\hat{x}_j(t_k) = a \times x_i(t_k) + b$ is equal to the sum of squares of the distances of the points in $\mathcal{P}$ from the line. Thus, minimizing the sse of the approximation is reduced to computing the *least squares regression line* for $\mathcal{P}$. The optimal values $a^*_{i,j}$ and $b^*_{i,j}$ follow from standard regression theory. ∎

In case $x_i(t)$ is constant (includes case n=1) the optimal values are $a^*_{i,j} = 0$ and $b^*_{i,j} = \frac{\sum_{1\leq k\leq n} x_j(t_k)}{n}$.

There is a vast literature on linear regression that can be of use for optimizing other error metrics such as relative or absolute error [4]. For brevity, in what follows we assume that the metric of interest is the sse.

Model-aware Cache Management. When a newer value of x_j is available the node stores it along with its current measurement x_i, if it is decided so by the cache admission policy. For a particular neighbor N_j or N_i, the cache-line for N_j (maintained by N_i) is a list of pairs of values of $x_i(t)$ and $x_j(t)$ collected at the same time. (There are many ways of optimizing representation of these lists that are outside the scope of this discussion.) Thus,

$$cache\_line(N_j) = \{(x_i(t_1), x_j(t_1)), (x_i(t_2), x_j(t_2)), \ldots\}$$

The number of pairs stored in a cache line will be decided dynamically and, in general, will be different per cache-line. Initially, when the cache is not full, new data values are stored in the corresponding cache lines. If the cache is full and a new measurement x_j is given, we weigh the gains of adding x_j in the cache, against the penalty of evicting a past measurement. We first list the actions we consider and then show how to choose among them.

- **Time-shift the cache-line for N_j:** We can remove the oldest pair $(x_i(t_1), x_j(t_1))$ from $cache\_line(N_j)$ and add a new entry $(x_i(t), x_j(t))$ (t=current time).
- **Augment the cache-line for N_j:** We may store the new pair $(x_i(t), x_j(t))$ in $cache\_line(N_j)$ and remove the *oldest* entry from another cache-line in order to keep the total space constant.
- **Leave as is:** We reject the latest measurement

Notice that victims are always chosen from the oldest member of a cache-line. The reason for this is twofold: we "force" the cache to gradually shift into newer observations and, at the same time, we are able to update the cache in linear time, as will be explained, instead of quadratic (had we chosen to consider every stored pair for eviction).

We now show how to evaluate the individual choices. For a cache-line c=$\{(x_i(t_1), x_j(t_1)), (x_i(t_2), x_j(t_2)), \ldots\}$ the average sse of the model $\hat{x}_j(t) = a \times x_i(t) + b$ over the values in c is

$$sse(c, a, b) = \frac{\sum_{t_k:(x_i(t_k),x_j(t_k))\in c}(x_j(t_k) - a \times x_i(t_k) - b)^2}{length(c)}$$

where $length(c)$ returns the number of pairs stored in the cache line. If no model were available, then node N_i could not provide an estimate for $x_j(t_k)$. The average sse of the no-answer policy is

$$no\_answer\_sse(c) = \frac{\sum_{t_k:(x_i(t_k),x_j(t_k))\in c} x_j^2(t_k)}{length(c)}$$

The expected benefit (over the no-answer policy) of using the model for estimating the $x_j(t_k)$ values in c is defined as

$$benefit(c, a, b) = no\_answer\_sse(c) - sse(c, a, b)$$

The cache admission and replacement strategy will be based on this benefit computation. Let $c_{shift} = \{(x_i(t_2), x_j(t_2)), \ldots, (x_i(t), x_j(t))\}$ be the resulting cache-line of the first option and let c_{aug} be the augmented cache line for N_j: $c_{aug} = c \cup \{(x_i(t), x_j(t))\}$. Let $a^*(c)$, $b^*(c)$ be the optimal model parameters for the values of c, see Lemma 1. We use the same notation for cache-lines c_{shift} and c_{aug}. The cache manager evaluates the benefits of each choice by performing the following tests in the order they are presented.

- If $benefit(c_{aug}, a^*(c), b^*(c)) \geq benefit(c_{aug}, a^*(c_{shift}), b^*(c_{shift}))$ and $benefit(c_{aug}, a^*(c), b^*(c)) \geq benefit(c_{aug}, a^*(c_{aug}), b^*(c_{aug}))$ then the model produced by the current state of the cache ($a^*(c), b^*(c)$) is more accurate than using the models computed from c_{shift} and c_{aug}, for evaluating all available observations of x_j. We therefore reject the new pair $(x_i(t), x_j(t))$.
- If $benefit(c_{aug}, a^*(c_{shift}), b^*(c_{shift})) \geq benefit(c_{aug}, a^*(c_{aug}), b^*(c_{aug}))$ we time-shift the cache-line of N_j and add the newest observation.

It should be noted that for N_j we compute the parameters a, b of a model (as is described in Lemma 1) but evaluate them using *all* known pairs (x_i, x_j), including the new observation $(x_i(t), x_j(t))$. This is why for N_j all benefits are computed on the values of c_{aug}, using the optimal parameters for each setup.

If both tests fail then $benefit(c_{aug}, a^*(c_{aug}), b^*(c_{aug}))$ is larger than $benefit(c_{aug}, a^*(c_{shift}), b^*(c_{shift}))$. Thus, extending the cache line of N_j with the new observation reduces the error. We will then try to find a victim from another cache line. The gains of augmenting cache line c of N_j, over time-shifting its values (that is equivalent to choosing the victim from c) is

$$Gain\_Augment_j = benefit(c_{aug}, a^*(c_{aug}), b^*(c_{aug})) -$$
$$benefit(c_{aug}, a^*(c_{shift}), b^*(c_{shift}))$$

For another cache line c' for node $N_k \neq N_j$, the penalty of evicting the oldest observation of c' is (c'' is c' minus its oldest pair)

$$Penalty\_Evict_k =$$
$$benefit(c', a^*(c'), b^*(c')) - benefit(c', a^*(c''), b^*(c''))$$

We will chose the cache line of the node for which the above penalty is the smallest, out of those in the cache for which $Penalty\_Evict_k < Gain\_Augment_j$ and evict its oldest pair. If no such victim is found but

$$benefit(c_{aug}, a^*(c_{shift}), b^*(c_{shift})) > benefit(c_{aug}, a^*(c), b^*(c))$$

time-shifting c is better than rejecting the new pair.

There is a final special case that is worth mentioning. If $x_j(t)$ is the first observation for N_j (i.e. c was empty), then the gain of augmenting c is $Gain\_Augment_j = x_j(t)^2$ that can be a large value. This may lead us evict a pair from another cache line. We thus need for a mechanism to prevent degradation of the models of other nodes because of a newly observed measurement for which we have no history and can not predict if it is worth trying to model. Thus, for *newcomers* we pick the victim in a round-robin fashion among all the available cache lines. This protects good models of small in amplitude measurements, when the available cache is too small.

The computation of the optimal $a^*(c)$, $b^*(c)$ parameters for a cache-line c takes time linear in the size of the cache-line. The same is true for the benefit computations. Thus, the overall time for this process is linear in the size of the cache. We can speed-up the computations by having pre-computed $Penalty\_Evect_k$ for all cache-lines in the cache. Notice that the benefit and model of c change only when an action is taken that affects c.

5. Discovery and Maintenance of Representative Nodes

The sensor nodes make use of the models they maintain in order to elect a smaller set of representative nodes in a localized fashion using a small (up to six) number of messages per node. The local policy that we employ for reducing the number of representatives is to have a node N_j choose as its representative the node N_i that can represent the larger number of nodes in its neighborhood. We then refine this selection in order to break ties and remove redundant representatives.

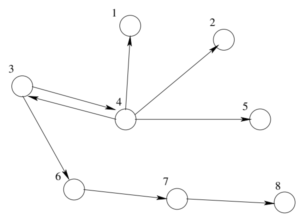

Figure 3. Initial election of representatives

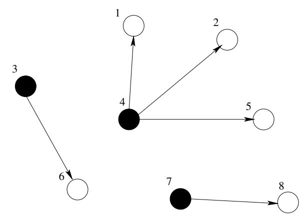

Figure 4. Final election (repr. in dark)

Each node has a status flag that is initially undefined and can take one of the following two values $ACTIVE$ or $PASSIVE$ interpreted as follows:

- An $ACTIVE$ node represents a non-empty set of nodes in the network that includes (by default) itself. Such a node responds to snapshot queries that involve any node it this set.
- A $PASSIVE$ node N_j is being represented by another node N_i in the network. Passive nodes do not respond to snapshot queries. In case of severe energy constraints, passive nodes may ask their representative to replace them on all user queries.

The first step is for each node N_j to broadcast to its neighbors a special message indicating that it is looking for representatives (*invitation phase*, see Table 2). This meassage also includes its current value $x_j(t)$. In turn, each node N_i that hears this message,[3] estimates, using its built-in model the value for $\hat{x}_j(t)$ and if $d(\hat{x}_j(t), x_j(t)) \leq T$, it adds N_j to a list $Cand\_nodes_i$ containing nodes it can represent.

The second step is for node N_i to broadcast list $Cand\_nodes_i$. Node N_j receives lists $Cand\_nodes_k$ from its neighbors and accepts as its representative the one, N_i, with the longest list. In case two nodes N_{i_1} and N_{i_2} can represent N_j and their lists are of the same length, we need for a way to break ties. Assuming that nodes have unique ids (that can be their MAC address), then without loss of generality a simple policy is for node N_j to pick the one with the largest id; i.e. favor N_{i_1} if $i_1 > i_2$. Finally, node N_j informs N_i that it accepts him as a representative.

We will demonstrate this process through a small example. We consider the nodes in Figure 3. Assume that the lists that the nodes broadcast in the second step are

$$\begin{array}{ll} Cand\_nodes_1 = \{N_2\} & Cand\_nodes_2 = \{\} \\ Cand\_nodes_3 = \{N_4, N_6\} & Cand\_nodes_4 = \{N_1, N_2, N_3, N_5\} \\ Cand\_nodes_5 = \{N_8\} & Cand\_nodes_6 = \{N_7\} \\ Cand\_nodes_7 = \{N_8\} & Cand\_nodes_8 = \{\} \end{array}$$

[3]This invitation message, as well as any other messages described below, are not propagated to the rest of the network. Thus, there is no retransmission cost. In addition we do not assume a reliable bidirectional communication protocol and messages may get lost.

```
// Rule-0: Break ties
IF ($N_i$ is represented by $N_j$) AND ($N_i$ represents $N_j$)
  IF (length($Cand\_nodes_i$)> length($Cand\_nodes_j$)) OR
     ((length($Cand\_nodes_i$)==length($Cand\_nodes_j$)) AND
     ($i > j$))
      SET $N_i$.mode=ACTIVE;
//Rule-1: Nodes that are not represented should stay active
IF ($N_i$ is represented by itself)
  SET $N_i$.mode=ACTIVE;
//Rule-2: Recall redundant representatives
IF ($N_i$.mode==ACTIVE) AND ($N_i$ is represented by another node $N_j$)
  Notify $N_j$ that it needs not represent $N_i$
//Rule-3: Requirement for PASSIVE mode
IF ($N_i$ is represented by another node $N_j$) AND
  ($N_i$ does not represent anyone)
    Notify $N_j$ to stay ACTIVE
    Wait for Acknowledgment from $N_j$[a]
    SET $N_i$.mode=PASSIVE;
//Rule-4: Clean up
IF ($time() > MAX\_WAIT$) AND ($N_i$.mode==UNDEFINED)
  IF ($rand() > P_{wait}$)
    SET $N_i$.mode=ACTIVE
  ELSE
    WAIT(1) //reconsider in next time unit
```

[a]Instead of sending individual acknowledgments, it is more efficient that node N_j broadcasts a single message indicating all nodes it is representing. Lost acknowledgments are handled by Rule-4.

Figure 5. Refining sel. of representatives

Consider node N_2. It has two offers from nodes N_1 and N_4. Since list $Cand\_nodes_4$ is longer than $Cand\_nodes_1$, it chooses N_4 as its representative. In case of node N_8, it breaks the tie between N_5 and N_7 by picking the one with the largest id. In Figure 3 we draw an arrow from node N_i to N_j, when node N_i is selected to represent N_j at the end of this step. In this example the initial set of representatives is N_3, N_4, N_6 and N_7. This set is further refined through an additional small round of negotiations among the nodes. During this phase each node will set its mode flag (that is initially undefined) to ACTIVE or PASSIVE. The logic that needs to be installed on every node N_i consists of the five "rules" shown in Figure 5.

Notice that all information required is already present at

Phase	#Messages (per node)	Description
Invitation	1	Each node broadcast its measurement $x_j(t)$
Model Evaluation	1	Node N_i compares $\hat{x}_j(t)$ against $x_j(t)$ Creates list of nodes it can represent $Cand\_nodes_i$. Broadcasts $Cand\_nodes_i$
Initial Selection	1	Node N_j picks N_i: $length(Cand\_nodes_i) = max_{k:N_j \in Cand\_nodes_k}(length(Cand\_nodes_k))$ N_j informs N_i that it (N_i) can represent N_j
Refinement	0-2	Implement Rules-0,1,2,3 and 4. Obtain final selection

Figure 2. Steps in representative discovery

the node. We explain this process in our running example: Because of Rule-0, node N_4 becomes ACTIVE. Because of Rule-2, node N_4 recalls its election of node N_3. In turn Node N_3 does not represent N_4 anymore. Because of Rule-3, nodes N_1, N_2, N_5 become PASSIVE. They also notify N_4 to stay ACTIVE (it is already so). Because of Rule-3, node N_8 becomes PASSIVE and informs N_7 to be ACTIVE. Because of Rule-2, node N_7 recalls its election of node N_6. Because of Rule-3, node N_6 becomes PASSIVE and informs N_3 to be ACTIVE. Finally, because of Rule-2, node N_3 recalls its election of node N_4.

The final set of representatives: N_3, N_4 and N_7 is shown in Figure 4. All other nodes are marked as PASSIVE. During this process, nodes do not change mode from PASSIVE to ACTIVE or vise versa. Thus, a node may send at most two messages because of Rule-2 or Rule-3. As shown in Table 2, the whole process requires at most five messages per node.

To handle link failures, or data dynamics that may lead to circular dependencies, nodes wait a maximum number of MAX_TIME time units to change their mode of operation from undefined to ACTIVE or PASSIVE. If this does not happen, then the node decides to stay ACTIVE. In order to avoid having all such nodes switching to ACTIVE mode simultaneously, a node that exceeded the MAX_TIME wait period, switches to ACTIVE mode with a probability P_{wait}, otherwise is will come back to the same rule in the next iteration.

5.1. Snapshot Maintenance

An ACTIVE node that only represents itself broadcasts periodically an invitation message to the network. Similarly, a PASSIVE node N_j that is represented by N_i sends a heartbeat message to its representative asking for its estimate $\hat{x}_j(t)$. If this estimate is out of bounds or node N_i does not respond (because of e.g. node failure) then N_j starts a new discovery process by inviting neighboring nodes to become its representative. It thus broadcasts an invitation

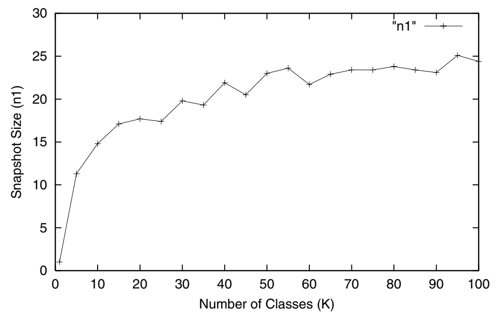

Figure 6. Varying number of classes

message, including value $x_j(t)$ and changes its mode to undefined. The nodes in the vicinity respond as is summarized in Table 2. The only modification to the protocol is that node N_j selects its representative by using the length of list $Cand\_nodes_i$ plus the number of nodes that N_i is already representing, which number is passed along with the list. By adding the heart-beat message by N_j and the response of its representative N_i, the maximum number of messages during maintenance is six (since $i \neq j$). In practice, the actual number of messages is smaller and depends, among other things, on node density.

Since representative nodes are handling snapshot queries, in addition to their regular workload, it is expected that their energy resources will be depleted faster. There are several ways to address this issue. A representative node N_i that finds its energy capacity fall below a threshold value, notifies the nodes it represents, who in-turn invite other nodes to represent them (N_i simply ignores these invitations). Another option is to use randomization in the selection of representatives, similar to the one used in the LEACH data routing protocol [8]. The key idea is to have a rotating set of representatives so that energy resources are drained uniformly. In Section 6.2 we compare a network setup that answers snapshot queries against a straightfor-

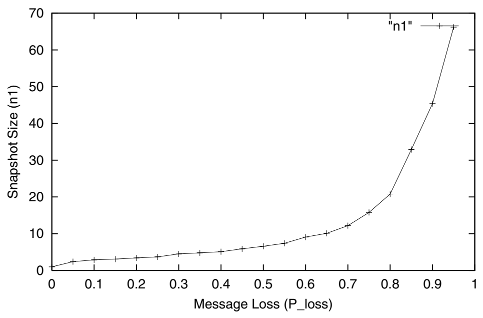

Figure 7. Varying message loss P_{loss}

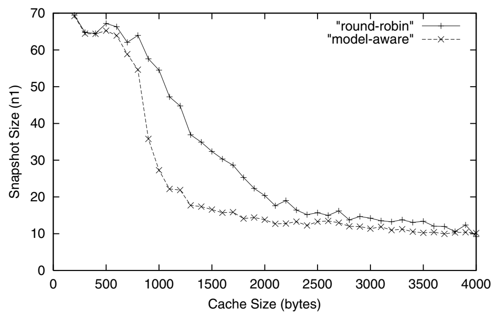

Figure 8. Varying cache size (K=10)

ward implementation that does not use representative nodes. We observe that even-though some background messaging and processing is required for the maintenance of the representative nodes, the lifetime of the network is substantially extended because most network nodes can stay idle during a snapshot query.

6. Experiments

We have developed a network simulator that allows as to vary several operational characteristics of the nodes like their location and transmission range, the probability of a link failure, the available memory etc. We first present a sensitivity analysis of our algorithms, varying various parameters. In section 6.2 we evaluate the savings, in energy consumption, obtained when using snapshot queries. Finally, in section 6.3 we show experiments with real weather data.

6.1. Sensitivity Analysis

We used synthetic data, similar in spirit with the one used in [5, 13]. The network consists of N=100 sensor nodes, randomly placed in a $[0 \dots 1) \times [0 \dots 1)$ two-dimensional area. For each node, we generated values following a random walk pattern, each with a randomly assigned step size in the range $(0 \dots 1]$. The initial value of each node was chosen uniformly in range $[0 \dots 1000)$. We then randomly partitioned the nodes into K classes. Nodes belonging to the same class i were making a random step (upwards or downwards) with the same probability $P_{move}[i]$. These probabilities were chosen uniformly in range $[0.2 \dots 1]$ (we excluded values less than 0.2 to make data more volatile).

In all runs we let the nodes operate for 100 time-units. During the first 10 time units, we executed a single query that was selecting the values from all nodes. This was done in order for the nodes to broadcast their values and let their neighbors build appropriate models. We then let the nodes silent for the next 90 time units and in the end we started a representative discovery phase for all nodes, using any cached values from the first 10% part.We repeated each experiment ten times and present the average values.

In the first experiment we used the following set-up. The communication range of the nodes was set to $\sqrt{2}$, allowing each node to listen to all messages in the $[0 \dots 1) \times [0 \dots 1)$ network area. Furthermore, the probability of message loss P_{loss} was zero meaning that network was modeled without any message loss (later-on we modify both parameters). The cache size on a node was fixed to 2,048 bytes. We used 4-byte floats for representing the values. Thus, a pair of values in a cache line was using 8-bytes. For a neighborhood of N=100 nodes, this means that, on the average, a node may cache between 2 and 3 measurements from another node in the network.

We varied the number of classes K between 1 and 100 and used a small error threshold value of T=1. In all runs, we used the sse as our error metric. In Figure 6 we plot the number of representatives (snapshot size) varying K using the aforementioned buffer size of 2,048 bytes. For K=1, when all nodes have the same behavior, the network successfully picks a single representative for all 100 nodes. When the number of classes exceeds 15, the number of representatives does not increase proportionally but stays within the range 17-25.

In Figure 7 we repeat the experiment when K=1, varying the probability of message loss P_{loss} from 0 to 0.95. When messages are lost during the first 10 time units, this affects the ability of a node to build accurate models for its neighbors. Message loss also affects the representative discovery phase, during which invitations, announcements of $Cand\_node$ lists and further negotiations require message exchange. For message loss of 30%, the network manages to find a representative set of size four, compared to one in the perfect communication case. Overall, message loss up to 80% does not significantly reduce the effectiveness of the models and the discovery process.

We varied the cache size allocated on a node from 200 bytes to 4KB when the number of classes K is 10. In Fig-

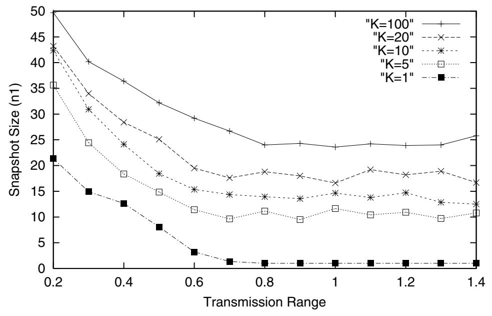

Figure 9. Varying transmission range

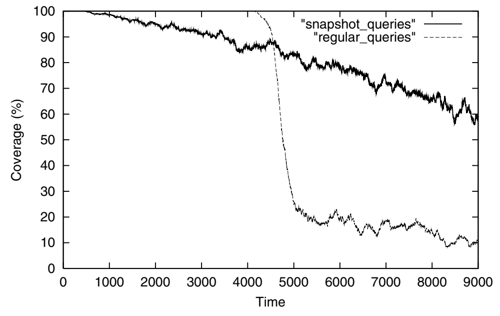

Figure 10. Network coverage overtime (K=T=1, transm. range=0.7)

	K=1		K=100	
	Transmission Range		Transmission Range	
Query Range	0.2	0.7	0.2	0.7
$W^2 = 0.01$	11%	29%	3%	7%
$W^2 = 0.1$	38%	77%	16%	24%
$W^2 = 0.5$	52%	91%	23%	49%

Table 3. Reduction in number of nodes participating in a spatial snapshot query

ure 8 we plot the number of representatives using the model-aware cache manager, versus a simpler round robin implementation. In this application, the access pattern is a series of "writes" (=updates to the models) ending with a single "read" (=discovery phase). Thus, the round-robin policy is equivalent to FIFO and LRU. For very small cache size, less than 500 bytes, there is no difference between the two methods. This is because for such small caches there is typically one pair per cache line and our algorithm falls back into using the round-robin policy, as described in Section 4. For a cache of 1,100 bytes, the number of representatives computed using the model-aware cache manager is less than half of that computed using the round-robin policy. When the cache is larger than 2.5KB, the differences are smaller, simply because for this data 2 or 3 pairs are enough for a fairly accurate model, as seen in Figure 6.

In Figure 9 we plot the number of representatives varying the transmission range of a node from 0.2 up to 1.4 for several values of K. For the chosen number of nodes (100), when the transmission range is less than 0.2, it often results in parts of the network being disconnected, which is unrealistic and can be correcting by adding more nodes. We notice that all lines become almost flat when the transmission range exceeds 0.7. This is because a range of $\sqrt{0.5} = 0.707$ is enough for sensor nodes located near the center of the $[0 \ldots 1) \times [0 \ldots 1)$ field to snoop on all messages in the network.

6.2. Savings During Snapshot Queries

We first experimented with aggregate queries over random parts of the network. For each query a *sink* node was chosen randomly. Then, using the flooding mechanism described in [11] an aggregation tree was formed, rooted at the sink node. The sensor nodes N_i whose measures were aggregated using that tree, were chosen using spatial predicate "$loc_i\ in\ [x - \frac{W}{2}, x + \frac{W}{2}] \times [y - \frac{W}{2}, y + \frac{W}{2}]$", where (x, y) is a random point in the $[0 \ldots 1) \times [0 \ldots 1)$ plane. This is a 2-dimensional range query of area equal to W^2.

We created a random set of 200 queries and executed each query in the set twice: once as a regular query and once as a snapshot query, see Section 3. We counted the number of nodes participated in each execution, denoted as $N_{regular}$ and $N_{snapshot}$ respectively. In Table 3 we show the savings $\frac{N_{regular} - N_{snapshot}}{N_{regular}}$ provided on the average by the snapshot queries (T=1). We note that when snapshot queries are used, a non-representative node may still be used for routing the aggregate and this is included in the numbers shown. For all runs, the aggregation tree was created using the vanilla method of [3, 11]. One can modify the protocol to favor (when applicable) representative nodes for routing the messages. This will result in further reduction in the number of sensor nodes used during snapshot queries than those presented in Table 3.

In Figure 10 we present an attempt to characterize the energy consumption in a network during regular and snapshot queries. The initial battery capacity of each node was set to be equal to the simulated cost of 500 transmissions. For accounting for energy consumption during computation, we modeled the processing cost of running the algorithm for maintaining the cache to be one tenth of the cost of transmitting a message. (This is probably an overestimate. For the slow-CPU Mica motes, the cost of sending one bit equals the cost of 1,000 operations [12]. For faster CPUs the ratio is a lot higher.) We then started executing random spatial

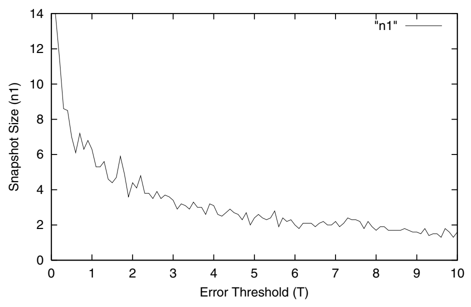

Figure 11. Varying the error threshold (T)

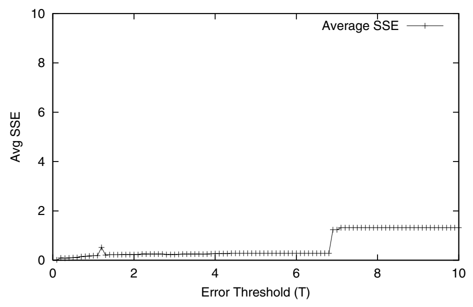

Figure 12. Avg sse of the approximation

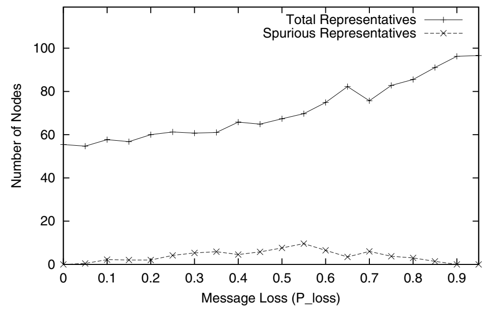

Figure 13. Spurious representatives

queries in the network, each of area 0.1. We made two runs. The first using regular queries and the second with snapshot queries. We executed our algorithms for electing and maintaining the representatives only during the second run. Thus, for regular queries, the only energy drain on a node was when responding to a query.

As nodes started draining their batteries and dying, we tracked the number of node measurements available to the query over the number of nodes that would have responded given infinite battery capacity. We call this metric *coverage*. For instance, if four nodes are within the spatial filter of the query and one of them has died, coverage is 75%. For the same query on the snapshot, the representative of the node that died might be available and in that case coverage will be 100%. In Figure 10 we notice that for snapshot queries coverage decreases gradually, mainly because representative nodes are drained faster as they handle a lot more queries. Coverage is a perfect 100% for regular queries up to the middle of the run but then it drops abruptly to less than 20%. This is because for non-snapshot queries energy drain is, roughly, uniform and there is a point when the network finds most of its nodes depleted. When snapshot queries are used, the aggregate energy consumption is substantially smaller. What is important is the area below each curve, which in the case of snapshot queries is significantly larger. In this run we used a simple maintenance protocol that replaced representative nodes as they died out. As discussed in Section 5.1, one can use randomization in the maintenance of the representative nodes (using ideas from [8]) to further improve network lifetime.

6.3. Experiments with Weather Data

For testing our framework on a realistic application we used weather data, providing wind speed measurements at a resolution of one minute for the year 2002. The data was collected at the weather station of the university of Washington.[4] We randomly chose 100 non-overlapping series of 100 values each from that dataset and used them in our simulation. The average value (over the 100 series) of the measurement was 5.8 and the average variance 2.8. We again used the first 10 values for training and ran the representative discovery phase after the last (100th) measurement. The cache size was 2,048 bytes and transmission range $\sqrt{2}$.

In Figure 11 we plot the average (over 10 repetitions) number of representatives $n1$ varying the error threshold T from 0.1 up to 10. Even for the smallest error threshold (that is substantially smaller than the variance of the data values) the models built by the sensors are accurate enough to result in a snapshot of 14% the size of the network. The average snapshot size drop quickly with increasing T, down to 1.5% when threshold is 10. In Figure 12 we plot the average sse in the estimates provided by the representatives for different values of T. We observe that the real error is in practice significantly smaller than the threshold used.

Figure 13 shows the number of spurious representatives resulting from message loss, when varying P_{loss} from 0 to 95% (T=0.1, transmission range=0.2). We also show the total number of representatives. Spurious representatives are the result of message loss in execution of Rule-2 (Figure 5). We notice that their number is very small and is in-fact decreasing for very high loss rates because most invitations are lost and fewer invocations of Rule-2 are encountered. Spurious representatives are easy to detect and can be auto-

[4]http://www-k12.atmos.washington.edu/k12

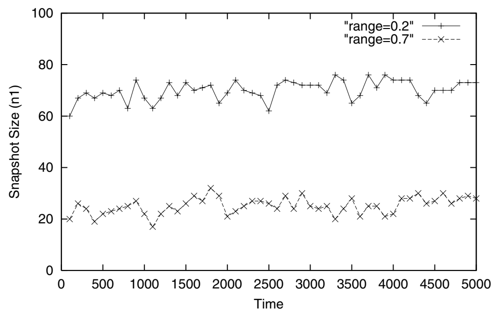

Figure 14. Number of representative nodes overtime

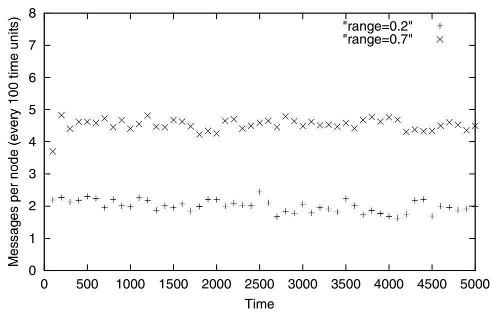

Figure 15. Number of messages during snapshot maintenance

matically corrected by the network (see Section 3).

For testing updates on the network snapshot we split the weather data into 100 series of 5,000 data values each. We updated the network snapshot, as explained in Section 5.1, every 100 time units. Between updates, we executed random queries and configured each node to snoop with probability 5% messages of its neighbors (at random intervals). In Figure 14 we plot the snapshot size overtime for two transmission ranges 0.2 and 0.7. For both ranges, it fluctuates slightly around the average number, 70 and 25 respectively. In Figure 15 we plot the average number of messages per node on each update. For the longer range (0.7) there are more messages, as more nodes respond to an invitation. The average number is 4.5 and 2 messages respectively, well below the maximum of six discussed in Section 5.1.

7. Conclusions

In this paper we described the operations of a *data-centric* network of sensor nodes that provides unambiguous data access for error tolerate applications. The network maintains a snapshot view, consisting of a small number of nodes, that captures the most characteristic features of the data observed. Applications can use the snapshot for their inquires reducing dramatically the aggregate energy drain. The snapshot is also useful as an alternative means of answering a posed query when nodes and network links fail. We presented a detailed experimental study of our framework and algorithms using synthetic and real data. Our algorithms are very robust and effective even when there is high message loss in the network and very limited resources at the nodes (memory, radio transmission range).

References

[1] M. Bawa, H. Garcia-Molina, A. Gionis and R. Motwani. Estimating Aggregates on a Peer-to-Peer Network. TR, 2003.

[2] A. Cerpa, D. Estrin. ASCENT: Adaptive Self-Configuring sEnsor Network Topologies. In *Proc. of INFOCOM*, 2002.

[3] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *ICDE*, 2004.

[4] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Compressing Historical Information in Sensor Networks. In *Proceedings of ACM SIGMOD Conference*, 2004.

[5] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Hierarchical In-Network Data Aggregation with Quality Guarantees. In *Proceedings of EDBT Conference*, 2004.

[6] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-Driven Data Acquisition in Sensor Networks. In *VLDB Conference*, pages 588–599, 2004.

[7] D. Estrin, R. Govindan, J. Heidermann, and S. Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. In *Proceedings of MobiCom*, 1999.

[8] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-Efficient Communication Protocol for Wireless Microsensor Networks. In *Proceedings of HICSS*, 2000.

[9] D. Kempe, A. Dobra, and J. Gehrke. Gossip-Based Computation of Aggregate Information. In *Proc. of FOCS*, 2003.

[10] S. Khanna and W. C. Tan. On Computing Functions with Uncertainty. In *ACM PODS Conference*, 2001.

[11] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: A Tiny Aggregation Service for ad hoc Sensor Networks. In *Proceedings of OSDI Conference*, 2002.

[12] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The Design of an Acquisitional Query processor for Sensor Networks. In *Proceedings of SIGMOD Conf.*, June 2003.

[13] C. Olston, J. Jiang, J. Widom. Adaptive Filters for Cont. Queries over Distributed Data Streams. In *SIGMOD*, 2003.

[14] C. Olston and J. Widom. Offering a Precision-Performance Tradeoff for Aggregation Queries over Replicated Data. In *Proceedings of VLDB Conference*, 2000.

[15] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The Cricket Location-Support System. In *MobiCom*, 2000.

[16] A. Sharaf, J. Beaver, A. Labrinidis, and P. Chrysanthis. Balancing Energy Efficiency and Quality of Aggregate Data in Sensor Networks. *VLDB Journal*, 2004.

[17] Y. Yao and J. Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. *SIGMOD Record*, 31(3):9–18, 2002.