Data Reduction Techniques in Sensor Networks

Antonios Deligiannakis
University of Maryland
adeli@cs.umd.edu

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

Abstract

Recent advances in microelectronics have made feasible the deployment of sensor networks for a variety of monitoring and surveillance tasks. The severe energy constraints met in such networks make imperative the design of energy efficient protocols for communication, which often constitutes the largest source of energy drain, of the collected information. In this paper, we describe several techniques that can be applied for the reduction of the transmitted information in different application scenarios.

1 Introduction

Recent advances in wireless technologies and microelectronics have made feasible, both from a technological as well as an economical point of view, the deployment of densely distributed sensor networks [11]. 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. In applications where sensors are powered by small batteries and replacing them is either too expensive or impossible (i.e., sensors thrown over a hostile environment), designing energy efficient protocols is essential to increase the lifetime of the sensor network. Since radio operation is by far the biggest factor of energy drain in sensor nodes [4], minimizing the number of transmissions is vital in data-centric applications. Even in the case when sensor nodes are attached to larger devices with ample power supply, reducing bandwidth consumption may still be important due to the wireless, multi-hop nature of communication and the short-range radios usually installed in the nodes.

Data-centric applications thus need to devise novel dissemination processes for minimizing the number of messages exchanged among the nodes. Nevertheless, in densely distributed sensor networks there is an abundance of information that can be collected. In order to minimize the volume of the transmitted data, we can apply two well known ideas: *aggregation* and *approximation*.

In-network aggregation is more suitable for exploratory, continuous queries that need to obtain a live estimate of some (aggregated) quantity. For example, sensors deployed in a metropolitan area can be used to obtain estimates on the number of observed vehicles. Temperature sensors in a warehouse can be used to keep track of average and maximum temperature for each floor of the building. Often, aggregated readings over a large number of sensors nodes show little variance, providing a great opportunity for reducing the number of (re)transmissions by the nodes when individual measurements change only slightly (as in temperature readings) or changes in measurements of neighboring nodes effectively cancel out (as in vehicle tracking).

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Approximation techniques, in the form of lossy data compression are more suitable for the collection of historical data through long-term queries. As an example, consider sensors dispersed in a wild forest, collecting meteorological measurements (such as pressure, humidity, temperature) for the purpose of obtaining a long term historical record and building models on the observed eco-system [8]. Each sensor generates a multi-valued data feed and often substantial compression can be achieved by exploiting natural correlations among these feeds (such as in case of pressure and humidity measurements). In such cases, sensor nodes are mostly "silent" (thus preserving energy) and periodically process and transmit large batches of their measurements to the monitoring station for further processing and archiving.

While the preferred method of data reduction, either by aggregation or by approximation, of the underlying measurements can be decided based on the application needs, there is a lot of room for optimization at the network level too. Sensor networks are inherently redundant; a typical employment uses a lot of redundant sensor nodes to cope with node or link failures [1]. Thus, extracting measurements from all nodes in the network for the purpose of answering a posed query may be both extremely expensive and unnecessary. In a data-centric network, nodes can coordinate with their neighbors and elect a small set of *representative nodes* among themselves, using a localized, data-driven bidding process [5]. These representative nodes constitute a *network snapshot* that can, in turn, answer posed queries while reducing substantially the energy consumption in the network. These nodes are also used as an alternative means of answering a posed query when nodes and network links fail, thus providing unambiguous data access to the applications.

2 Characteristics of Sensor Nodes

Depending on the targeted application, sensor nodes with widely different characteristics may be used. Even though the processing and memory capabilities of sensor nodes are still limited, in recent years they have increased at a rate similar to Moore's law. On the other hand, the amount of energy stored in the batteries used in such nodes has exhibited a mere 2-3% annual growth.[1] Since replacing the sensor batteries may be very expensive, and often impossible due to their unattended deployment, unless the sensors are attached to and powered by a larger unit, designing energy-efficient protocols is essential to increase the lifetime of the sensor network.

The actual energy consumption by each sensor node depends on its current state. In general, each sensor node can be in one of the following states: 1) *low-duty cycle*, where the sensor is in sleep mode and a minimal amount of energy is consumed; 2) *idle listening*, where the sensor node is listening for possible data intended for it; 3) *processing*, where the node performs computation based on its obtained measurements and its received data; and, 4) *receiving/transmitting*, where the node either receives or transmits data or control messages.

The cost of processing can be significant but is generally much lower than the cost of transmission. For example, in the Berkeley MICA nodes sending one bit of data costs as much energy as 1,000 CPU instructions [7]. For long-distance radios, the transmission cost dominates the receiving and idle listening costs. For short-range radios, these costs are comparable. For instance, in the Berkeley MICA2 motes the power consumption ratio of transmitting/receiving at 433MHz with RF signal power of 1mW is 1.41:1 [13], while this ratio can become even larger than 3:1 for the same type of sensor when the radio transmission power is increased [10]. To increase the lifetime of the network, some common goals of sensor network applications are (in order of importance) to maximize the time when a node is in a low-duty cycle, to reduce the amount of transmitted and received data, and to reduce the idle listening time. We note here that reducing the size of the transmitted data results in multiple benefits, since this also corresponds to a reduction of not only control messages, but also leads to fewer message collisions and retransmissions. Moreover, nodes that refrain from transmitting messages may switch to the low-duty cycle mode faster, therefore further reducing their energy drain.

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

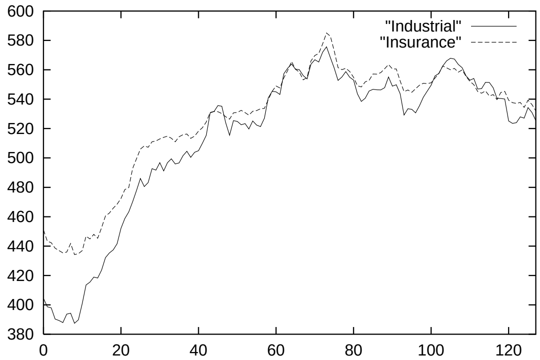

Figure 1: Example of two correlated signals (Stock Market)

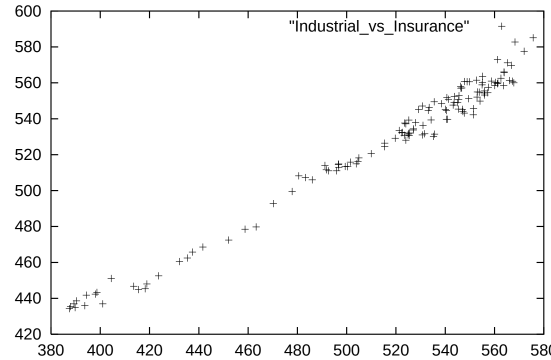

Figure 2: XY scatter plot of Industrial (X axis) vs Insurance (Y axis)

3 A Lossy Compression Framework for Historical Measurements

Many real signals observed by the sensors, such as temperature, dew-point, pressure etc. are naturally correlated. The same is often true in other domains. For instance, stock market indexes or foreign currencies often exhibit strong correlations. In Figure 1 we plot the average Industrial and Insurance indexes from the New York stock market for 128 consecutive days.[2] Both indexes show similar fluctuations, a clear sign of strong correlation. Figure 2 depicts a XY scatter plot of the same values. This plot is created by pairing values of the Industrial (X-coordinate) and Insurance (Y-coordinate) indexes of the same day and plotting these points in a two-dimensional plane. The strong correlation among these values makes most points lie on a straight line. This observation suggests the following compression scheme, inspired from regression theory. Assuming that the Industrial index (call it $\vec{X}$) is given to us in a time-series of 128 values, we can approximate the other time-series (Insurance: $\vec{Y}$) as $\vec{Y'} = a * \vec{X} + b$. The coefficients a and b are determined by the condition that the sum of the square residuals, or equivalently the L_2 error norm $||\vec{Y'} - \vec{Y}||_2$, is minimized. This is nothing more than standard linear regression. However, unlike previous methods, we will not attempt to approximate each time-series independently using regression. In Figure 1 we see that the series themselves are not linear, i.e., they would be poorly approximated with a linear model. Instead, we will use regression to approximate piece-wise correlations of each series to a base signal $\vec{X}$ that we will choose accordingly. In the example of Figure 2, the base signal can be the Industrial index ($\vec{X}$) and the approximation of the Insurance index will be just two values (a, b). In practice the base signal will be much smaller than the complete time series, since it only needs to capture the "important" trends of the target signal $\vec{Y}$. For instance, in case $\vec{Y}$ is periodic, a sample of the period would suffice.

The SBR framework. In the general case, each sensor monitors N distinct quantities $\vec{Y_i}$, $1 \leq i \leq N$. Without loss of generality we assume that measurements are sampled with the same rate. When enough data is collected (for instance, when the sensor memory buffers become full), the latest NxM values are processed and each row i (of length M) is approximated by a much smaller set of B_i values, i.e. $B_i \ll M$. The resulting "compressed" representation, of total size equal to $\sum_{i=1}^{N} B_i$, is then transmitted to the base station. The base station maintains the data in this compact representation by appending the latest "chunk" to a log file. A separate file exists for each sensor that is in contact with the base station. This process is illustrated in Figure 3. Each sensor allocates a small amount of memory of size M_{base} for what we call the *base signal*. This is a compact ordered collection of values of prominent features that we extract from the recorded values and are used as a base reference in the approximate representation that is transmitted to the base station. The data values that the sensor transmits to the base station are encoded using the in-memory values of the base signal at the time of the transmission. The base

[2] Data at `http://www.marketdata.nasdaq.com/mr4b.html`.

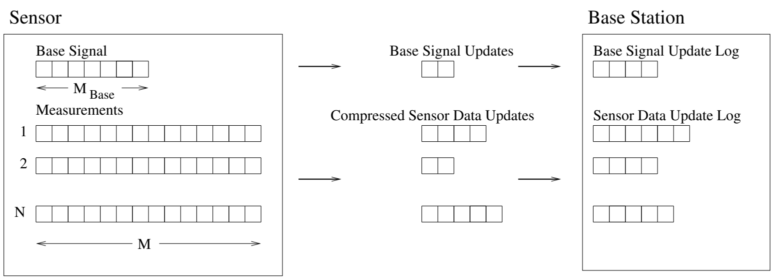

Figure 3: Transfer of approximate data values and of the base signal from each sensor to the base station

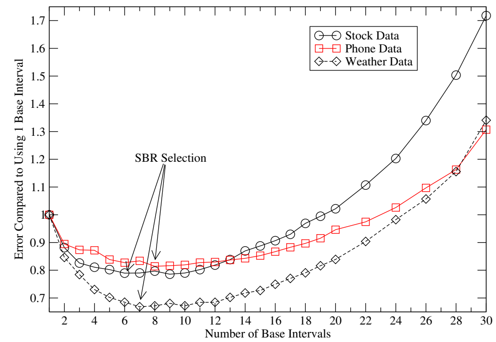

Figure 4: SSE error vs base signal size

signal may be updated at each transmission to ensure that it will be able to capture newly observed data features and that the obtained approximation will be of good quality. When such updates occur, they are transmitted along with the data values and appended in a special log file that is unique for each sensor.

The Self-Based Regression algorithm (SBR) breaks the data intervals $\vec{Y_i}$ into smaller data segments $I_i[k..l] = (Y_i[k], \ldots Y_i[l])$ and then pairs each one to an interval of the base signal of equal length. As discussed below, the base signal is simply the concatenation of several intervals of the same length W extracted from the data. The data interval I_i is shifted over the base signal and at each position s we compute the regression parameters for the approximation $\hat{I}_i[j] = a \times X[s + j - k] + b$, $k \leq j \leq l$ and retain the shift value $s = s^*$ that minimizes the sum-squared error of the approximation. The algorithm starts with a single data interval for each row of the collected data (Y_i). In each iteration, the interval with the largest error in the approximation is selected and divided in two halves. The compressed representation of a data interval $I_i[k..l]$ consists of four values: the shift position s^* that minimizes the error of the approximation, the two regression parameters a,b and the start of the data interval k in Y_i. The base station will sort the intervals based on their start position and, thus, there is no need to transmit their ending position. Given a target budget B (size of compressed data) we can use at most $B/4$ intervals using this representation.

Base Signal Construction. We can think of the base signal as a dictionary of features used to describe the data values. The richer the pool of features we store in the base signal the better the approximation. On the other hand, these features have to be (i) kept in the memory of the sensor to be used as a reference by the data-reduction algorithm and (ii) sent to the base station in order for it to be able to reconstruct the values. Thus, for a target bandwidth constraint B (number of values that can be transmitted), performing more insert and update operations on the base signal implies less bandwidth remaining for approximating the data values, and, thus, fewer data intervals that can be obtained from the recursive process described above.

We can avoid the need of transmitting the base signal by agreeing a-priori on a set of functions that will be used in the regression process. For instance, a set of cosine functions (as in the Distinct Cosine Transform) can be used for constructing a "virtual" base signal that does not need to be communicated. Similarly, using the identity function $X[i] = i$ reduces the compression algorithm to standard linear regression of each data interval. However, such an approach makes assumptions that may not hold for the data at hand. In [2] we have proposed a process for generating the base signal from the data values. The key idea is to break the measurements into intervals of the same length W. Each interval (termed *candidate base interval*) is assigned a score based on the reduction in the error of the approximation obtained by adding the interval to the base signal. Using a greedy algorithm we can select the top-few candidate intervals, up to the amount of available memory M_{base}. Then a binary-search process is used to eventually decide how many of those candidate intervals need to be retained.

The search space is illustrated in Figure 4 for three real data sets, discussed in [2]. The figure plots the error of only the initial transmission as the size of the base signal is varied, manually, from 1 to 30 intervals. We further show the selection of the binary-search process. For presentation purposes, the errors for each data set

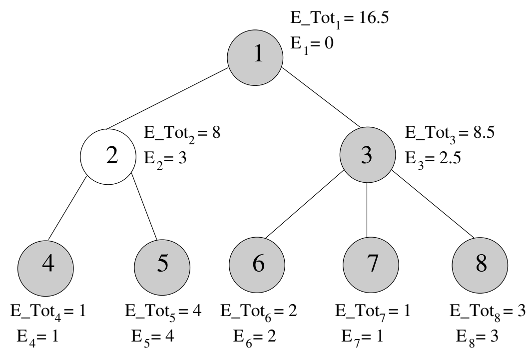

Figure 5: Error Filters on Aggregation Tree

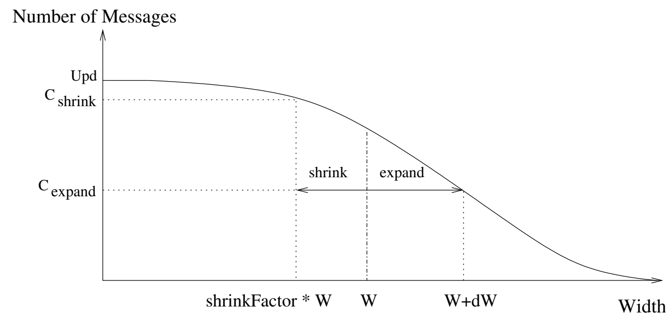

Figure 6: Potential Gain of a Node

have been divided by the error of the approximation when using just one interval. We notice that initially, by adding more candidate intervals to the base signal the error of the approximation is reduced. However, after a point, adding more intervals that need to be transmitted to the base station leaves insufficient budget for the recursive process that splits the data, and thus, the error of the approximation is eventually increased.

For a data set containing $n = N \times M$ measurements to approximate the complete SBR algorithm takes $O(n^{1.5})$ time and requires linear space, while its running time scales linearly to the size of both the transmitted data and the base signal. The algorithm is invoked periodically, when enough data has been collected. Moreover, our work [2] has demonstrated that, after the initial transmissions, the base signal is already of good quality and few intervals are inserted in it. This provides us with the choice not to update the base signal in many subsequent invocations, thus reducing the algorithm's running time, in these cases, to only a linear dependency on n.

4 Approximate in-Network Data Aggregation

In-Network Data Aggregation. The data aggregation process in sensor networks consists of several steps. First, the posed query is disseminated through the network, in search of nodes collecting data relevant to the query. Each sensor then selects one the nodes through which it received the announcement as its *parent node*. The resulting structure is often referred to as the *aggregation tree*. Non-leaf nodes of that tree aggregate the values of their children before transmitting the aggregate result to their parents. In [6], after the aggregation tree has been created, the nodes carefully schedule the periods when they transmit and receive data. The idea is for a parent node to be listening for values from its child nodes within specific intervals of each *epoch* and then transmit upwards a single partial aggregate for the whole subtree.

Framework Description. In order to limit the number of transmitted messages and, thus, the energy consumption in sensor networks during the computation of continuous aggregate queries, our algorithms install error filters on the nodes of the aggregation tree.[3] Each node N_i transmits the partial aggregate that it computes at each epoch for its subtree only if this value deviates by more than the maximum error E_i of the node's filter from the last transmitted partial aggregate. This method allows nodes to refrain from transmitting messages about small changes in the value of their partial aggregate. The E_i values determine the maximum deviation E_Tot_i of the reported from the true aggregate value at each node. For example, for the SUM aggregate, this deviation at the monitoring node is upper bounded (ignoring message losses) by: $\sum_i E_i$. A sample aggregation tree is depicted in Figure 5. As our work [3] has demonstrated, allowing a small error in the reported aggregate can lead to dramatic bandwidth (and thus energy) savings. The challenge is, of course, given the maximum error tolerated by the monitoring node, to calculate and periodically adjust the node filters in order to minimize the

[3]The use of error filters in error-tolerate applications in flat, non-hierarchical domains has been investigated in [9].

bandwidth consumption.

Algorithmic Challenges. When designing adaptive algorithms for in-network data aggregation in sensor networks, one has to keep in mind several challenges/goals. First, communicating *individual* node statistics is very expensive, since this information cannot be aggregated inside the tree, and may outweigh the benefits of approximate data aggregation, namely the reduction in the size of the transmitted data. Thus, our algorithm should not try to estimate the number of messages generated by each node's transmissions, since this depends on where this message is aggregated with messages from other nodes. Second, the error budget should be distributed to the nodes that are expected to reduce their bandwidth consumption the most by such a process. This benefit depends on neither the magnitude of the partial aggregate values of the node nor the node's number of transmissions over a period, but on the magnitude of the changes on the calculated partial aggregate. Isolating nodes with large variance on their partial aggregate and redistributing their error to other nodes is crucial for the effectiveness of our algorithm [3]. Finally, in the case of nodes where the transmitted differences from their children often result in small changes on the partial aggregate value, our algorithm should be able to identify this fact. We deal with this latter challenge by applying the error filters on the calculated partial aggregate values and not on each node's individual measurements. For the first two challenges, we collect a set of easily computed and composable statistics at each node. These statistics are used for the periodic adjustment of the error filters.

Algorithm Overview. Every Upd epochs all nodes shrink the widths of their filters by a shrinking factor $shrinkFactor$ ($0 < shrinkFactor < 1$). After this process, the monitoring node has an error budget of size $E\_Global \times (1 - shrinkFactor)$, where E_Global is the maximum error of the application, that it can redistribute recursively to the nodes of the network. Each node, between two consecutive update periods, calculates its *potential gain* as follows: At each epoch the node keeps track of the number of transmissions C_{shrink} that it would have performed with the default (smaller) filter at the next update period of width $shrinkFactor \times W_i$, where $W_i = 2 \times E_i$. The node also calculates the corresponding number of transmissions C_{expand} with a larger filter of width $W_i + dW$ and sets its potential gain to $Gain_i = C_{shrink} - C_{expand}$. This process is illustrated in Figure 6. The *cumulative gain* of a node's subtree is then calculated as the sum of the cumulative gains of the node's children and the node's potential gain. This requires only the transmission of the cumulative gains (a single value for each node) at the last epoch before the new update period. The available error budget is then distributed top-down proportionally, at each node, to each subtree's cumulative gain. In this process, nodes that exhibit large variance in their partial aggregate values will exhibit small potential gains and, thus, their error will gradually shrink and be redistributed to nodes that will benefit from an increase in their error filter.

We note here that the dual problem of minimizing the application maximum error given a bandwidth constraint can also be solved in a similar manner, using statistics collected at each node. The problem is more complicated, though, because the controlled quantity at each node (the width of its error filter) is different from the monitored quantity (the bandwidth consumption) and the bandwidth needs to be carefully computed, monitored and then disseminated amongst the sensor nodes.

5 Design of Data-Centric Sensor Networks

Sensor networks are inherently dynamic. Such networks must adapt to a wide variety of challenges imposed by the uncontrolled environment in which they operate. As nodes become cheaper to manufacture and operate, one way of addressing the challenges imposed on unattended networks is redundancy [1]. Redundant nodes ensure network coverage in regions with non-uniform communication density due to environmental dynamics. Redundancy further increases the amount of data that can be mined in large-scale networks.

Writing data-driven applications in such a dynamic environment can be daunting. Again, the major challenge is to design localized algorithms that will perform most of the processing in the network itself in order to reduce traffic and, thus, preserve energy. Instead of designing database applications that need to hassle with low-level networking details, we envision the use of data-centric networks that allow transparent access to the collected measurements in a unified way. For instance, when queried nodes fail, the network should self-configure to use redundant stand-by nodes as in [4], under the condition that the new nodes contain fairly similar measurements, where similarity needs to be quantified in an application-meaningful way [5]. This can be achieved using a localized mode of operation in which nodes can coordinate with their neighbors and elect a small set of *representative nodes* among themselves. Such a set of representatives, termed *network snapshot* [5], has many advantages. The location and measurements of the representative nodes provide a picture of the value distribution in the network. Furthermore, the network snapshot can be used for answering user queries in a more energy-efficient way, since significantly fewer nodes need to be involved in query processing. Finally, an elected representative node can take over for another node in its vicinity that may have failed or is temporarily out of reach.

6 Conclusions and Future Directions

In this paper we described several techniques for the reduction of the transmitted data in several sensor network applications, ranging from the communication of historical measurements to answering approximate aggregate continuous and snapshot queries. While these techniques aim to prolong the lifetime of the network, there are several issues that need to be additionally addressed. Little work has been done on the optimization of multiple concurrent continuous queries over sensor networks. The work of Olston et al. in [9] may provide some helpful solutions in this area. Moreover, in the presence of nodes with different transmission frequencies, as in the case of approximate aggregate query processing, several communication and synchronization algorithms may need to be revisited [12]. For example, the selection of the aggregation tree is often performed by assuming equal frequency of transmissions by all nodes. However, it might be more beneficial to prevent nodes that exhibit large variance in their measurements from appearing in lower levels of the tree, since such nodes often trigger transmissions on their ancestors as well. Such optimizations may lead to even larger energy savings.

References

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

[2] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos. Compressing Historical Information in Sensor Networks. In *SIGMOD*, 2004.

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

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

[5] Y. Kotidis. Snapshot Queries: Towards Data-Centric Sensor Networks. In *Proceedings of ICDE*, 2005.

[6] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A Tiny Aggregation Service for ad hoc Sensor Networks. In *OSDI Conf.*, 2002.

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

[8] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *WSNA 2002*, pages 88–97, 2002.

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

[10] V. Shnayder, M. Hempstead, B. Chen, G. W. Allen, and M.. Welsh. Simulating the Power Consumption of Large-Scale Sensor Network Applications. In *Sensys*, 2004.

[11] B. Warneke, M. Last, B. Liebowitz, and K. S.J. Pister. Smart Dust: Communicating with a Cubic-Millimeter Computer. *IEEE Computer*, 34(1):44–51, 2001.

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

[13] W. Ye and J. Heidermann. Medium Access Control in Wireless Sensor Networks. Technical report, USC/ISI, 2003.