A Distributed and Adaptive Signal Processing Approach to Reducing Energy Consumption in Sensor Networks

Jim Chou
Department of EECS
University of California at Berkeley
Berkeley, CA 94709
Email: jimchou@eecs.berkeley.edu

Dragan Petrovic
Department of EECS
University of California at Berkeley
Berkeley, CA 94709
Email: dragan@eecs.berkeley.edu

Kannan Ramchandran
Department of EECS
University of California at Berkeley
Berkeley, CA 94709
Email: kannanr@eecs.berkeley.edu

***Abstract*— We propose a novel approach to reducing energy consumption in sensor networks using a distributed adaptive signal processing framework and efficient algorithm [1]. While the topic of energy-aware routing to alleviate energy consumption in sensor networks has received attention recently [1,2], in this paper, we propose an orthogonal approach to previous methods. Specifically, we propose a distributed way of continuously exploiting existing correlations in sensor data based on adaptive signal processing and distributed source coding principles. Our approach enables sensor nodes to blindly compress their readings with respect to one another without the need for explicit and energy-expensive inter-sensor communication to effect this compression. Furthermore, the distributed algorithm used by each sensor node is extremely low in complexity and easy to implement (i.e., one modulo operation), while an adaptive filtering framework is used at the data gathering unit to continuously learn the relevant correlation structures in the sensor data. Our simulations show the power of our proposed algorithms, revealing their potential to effect significant energy savings (from 10%-65%) for typical sensor data corresponding to a multitude of sensor modalities.**

I. Introduction

advances in wireless networking and embedded microprocessor designs have enabled the creation of dense low-power sensor networks. These sensor networks consist of nodes endowed with a multitude of sensing modalities such as temperature, pressure, light, magnetometer, infrared, audio, video, etc. The nodes are typically of small physical dimensions and operated by battery power, making energy consumption a major concern. For example, failure of a set of nodes in the sensor network due to energy depletion can lead to a partition of the sensor network and loss of potentially critical information. Motivated by this, there has been considerable recent interest in the area of energy-aware routing for ad hoc and sensor networks [1], [2], [3] and efficient information processing [4], [5] to reduce the energy usage of sensor nodes. For example, one method of conserving energy in a sensor node is to aggregate packets along the sensor paths to reduce header overhead. In this paper, we propose a new method of conserving energy in sensor networks that is mutually exclusive to the above approaches, and can be used in combination with them to increase energy reduction.

Our approach is based on judiciously exploiting existing sensor data correlations in a distributed manner. Correlations in sensor data are brought about by the spatio-temporal characteristics of the physical medium being sensed. Dense sensor networks are particularly rich in correlations, where spatially dense nodes are typically needed to acquire fine spatial resolution in the data being sensed, and for fault tolerance from individual node failures. Examples of correlated sensors include temperature and humidity sensors in a similar geographic region, or magnetometric sensors tracking a moving vehicle. Another interesting example of correlated sensor data involves audio field sensors (microphones) that sense a common event such as a concert or whale cries. Audio data is particularly interesting in that it is rich in spatial correlation structure due to the presence of echoes, causing multiple sensors to pick up attenuated and delayed versions of a common sound origin.

We propose to remove the redundancy caused by these inherent correlations in the sensor data through a distributed compression algorithm which obviates the need for the sensors to exchange their data among each other in order to strip their common redundancy. Rather surprisingly, we will show that compression can be effected in a fully blind manner without the sensor nodes ever knowing what the other correlated sensor nodes have measured. Our proposed paradigm is particularly effective for sensor network architectures having two types of nodes: sensing nodes and data-gathering nodes. The sensing nodes gather data of a specific type and transmit this data upon being queried. The data gathering node queries specific sensors in order to gather information in which it is interested (see Fig. 1). We will assume the above architecture (Fig. 1) for the rest of the paper and show that for such an architecture, we can devise compression algorithms that have *very lightweight encoders, yet can achieve significant savings*. Note, that we target very lightweight encoders in this paper because we assume that the sensors have limited

[1]This work was supported in part by DARPA-F30602-00-2-0538, NSF-CCR-0219722 and Intel.

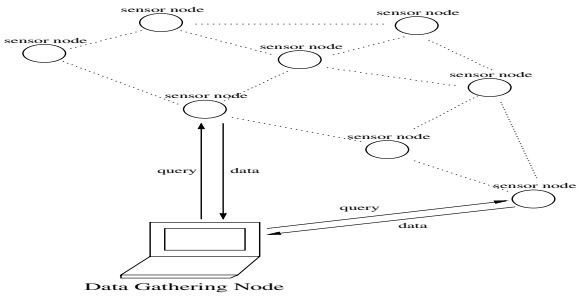

Fig. 1. An example sensor network: a computer acts as the data gathering node, and queries various sensors to collect data

compute power, but the constructions introduced in this paper can be easily strengthened given greater compute power at the sensors. The savings are achieved by having the data gathering node track the correlation structure among nodes and then use this information to effect distributed sensor data compression. The correlation structure is determined by using an adaptive prediction algorithm. *The sensors, however, do not need to know the correlation structure; they need to know only the number of bits that they should use for encoding their measurements.* As a result, each sensor node is required to perform very few operations in order to encode its data. The decoder, however, is considerably more complex, but it resides on the data gathering node, which is not assumed to be energy constrained. Preliminary results based on our distributed compression and adaptive prediction algorithms perform well in realistic scenarios, achieving 10-65% energy savings for each sensor in typical cases. In addition, our distributed compression architecture can be combined with other energy saving methods such as packet/data aggregation to achieve further gains.

Two of the main challenges in designing a system as described above include (1) devising a computationally inexpensive encoder that can support multiple compression rates and (2) determining an adaptive correlation-tracking algorithm that can continuously track the amount of correlation that exists between the sensor nodes. We will look at the above two issues in the following sections. In the next section, we start by devising a computationally inexpensive compression algorithm for the sensor nodes. In section 3, we will present the correlation tracking algorithm. In section 4, we will integrate the above components into a complete system. Simulation results are given in section 5 and we conclude with some remarks in section 6.

II. Distributed compression

The appeal of using distributed compression lies in the fact that each sensor can compress its data without knowing what the other sensors are measuring. In fact, the sensors do not even need to know the correlation structure between its data and that of the other sensors. As a result, an end-to-end compression system that achieves a significant savings across the network can be built, where the endpoints consist of the sensor node and the data gathering node.

To build a distributed compression system, we propose to use an asymmetric coding method among the sensors. Specifically, we propose to build upon the architecture of Fig. 2 which is designed for two nodes. In Fig. 2, there are two nodes, each of which measures data using an Analog-to-Digital (A/D) converter. One of the sensor nodes will either transmit its data Y directly to the data gathering node or compress its readings with respect to its own previous readings while the other sensor node compresses its data X with respect to its own previous readings *and* readings from other sensors and then transmits the compressed data m to the data gathering node. The decoder will then try to decode m to X, given that Y is correlated to X. In the discrete alphabet case, it can be shown that the compression performance of the above architecture can match the case where Y is available to the sensor node that is measuring X.

To extend the above architecture (Fig. 2) to n nodes we will have one node send its data either uncoded (i.e., Y) or compressed with respect to its past. The data gathering node can decode this reading without receiving anything from the other sensors. The other sensors can compress their data with respect to Y, without even knowing their correlation structure with respect to Y. The data gathering node will keep track of the correlation structure and inform the sensors of the number of bits that they shall use for encoding. In the compression literature, Y is often referred to as side-information and the above architectures are often referred to as compression with side information [6].

To develop code constructions for distributed compression, we will start by giving some background information on source coding with side information and then introduce a code construction that achieves good performance at a low encoding cost.

A. Background on compression with side information

In 1973, Slepian and Wolf presented a suprising result to the source coding (compression) community [6]. The result states that if two discrete alphabet random variables X and Y are correlated according to some arbitrary probability distribution $p(x, y)$, then X can be compressed without access to Y without losing any compression performance with respect to the case where X is compressed with access to Y. More formally, without having access to Y, X can be compressed using $H(X|Y)$ bits where

$$H(X|Y) = -\sum_{y} P_Y(y) \sum_{x} P_X(x|y) log_2 P_X(x|y) \qquad (1)$$

The quantity, $H(X|Y)$ is often interpreted as the "uncertainty" remaining in the random variable X given the observation of Y [7]. This is the same compression performance that would be achieved if X were compressed while having access to Y. To provide the intuition behind this result, we provide the following example.

Example 1: Consider X and Y to be equiprobable 3-bit data sets which are correlated in the following way: $d_H(X, Y) \leq 1$,

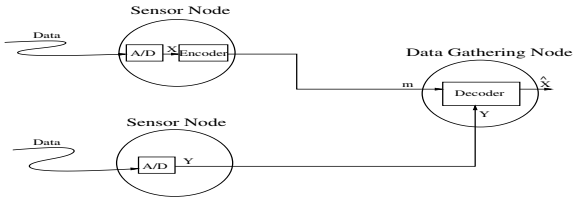

Fig. 2. Distributed compression: the encoder compresses X given that the decoder has access to Y, which is correlated to X.

where $d_H(.,.)$ denotes Hamming distance. When Y is known both at the encoder and decoder, we can compress X to 2 bits, conveying the information about the uncertainty of X given Y (i.e., the modulo-two sum of X and Y given by: (000),(001),(010) and (100)). Now, if Y is known only at the decoder, we can surprisingly still compress X to 2 bits. The method of construction stems from the following argument: if the decoder knows that X=000 or X=111, then it is wasteful to spend any bits to differentiate between the two. In fact, we can group X=000 and X=111 into one coset (it is exactly the so-called principal coset of the length-3 repetition code). In a similar fashion, we can partition the remaining space of 3-bit binary codewords into 3 different cosets with each coset containing the original codewords offset by a unique and correctable error pattern. Since there are 4 cosets, we need to spend only 2 bits to specify the coset in which X belongs. The four cosets are given as

$$\text{coset-1} = (000, 111), \quad \text{coset-2} = (001, 110),$$
$$\text{coset-3} = (010, 101), \quad \text{coset-4} = (011, 110)$$

The decoder can recover X perfectly by decoding Y to the closest (in hamming distance) codeword in the coset specified by the encoder. Thus the encoder does not need to know the realization of Y for optimal encoding.

The above results were established only for discrete random variables. In 1976, Wyner and Ziv extended the results of [6] to lossy distributed compression by proving that under certain conditions [8], there are no performance degradations for lossy compression with side information available at the decoder as compared to lossy compression with side information available at both the encoder and decoder.

The results established by [6] and [8] are theoretical results, however, and as a result do not provide intuition as to how one might achieve the predicted theoretical bounds practically. In 1999, Pradhan and Ramchandran [9] prescribed practical constructions for distributed compression in an attempt to achieve the bounds predicted by [6] and [8]. The resulting codes perform well, but cannot be directly used for sensor networks because they are not designed to support different compression rates. To achieve distributed compression in a sensor network, it is desirable to have one underlying codebook that is not changed among the sensors but can also support multiple compression rates. The reason for needing a codebook that supports multiple compression rates is that the compression rate is directly dependent on the amount of correlation in the data, which might be time-varying. Motivated by the above, we have devised a tree-based distributed compression code that can provide variable-rate compression *without the need for changing the underlying codebook.*

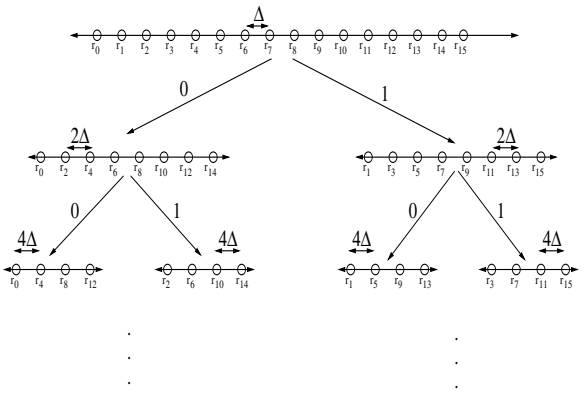

Fig. 3. A tree-based contruction for compression with side information. The root of the tree contains 2^4 values, and two partitions of the root quantizer are shown.

B. Code construction

In this section we propose a codebook construction that will allow an encoder to encode a random variable X given that the decoder has access to a correlated random variable Y. This construction can then be applied to a sensor network as shown in Fig. 2. The main design goal of our code construction is to support multiple compression rates, in addition to being computationally inexpensive. In support of our goal of minimizing the computations for each sensor node, we will not be looking into code constructions that use complicated error correction codes. Error correction codes, can however, be easily incorporated into our construction but will lead to more complexity for each sensor node. Our uncoded code construction is as follows. We start with a root codebook that contains 2^n representative values on the real axis. We then partition the root codebook into two subsets consisting of the even-indexed representations and the odd-indexed representations. We represent these two sub-codebooks as children nodes of the root codebook. We further partition each of these nodes into sub-codebooks and represent them as children nodes in the second level of the tree structure. This process is repeated n times, resulting in an n-level tree structure that contains 2^n leaf nodes, each of which represents a subcodebook that contains one of the original 2^n values. An example partition is given in Fig. 3, where we use $n = 4$ and show only 2 levels of the partition. Note from this tree-based codebook construction that if the spacing between representative values is denoted by Δ, then each of the subcodebooks at level-i in the tree will contain representative values that are spaced apart by $2^i\Delta$. In a sensor network, a reading will typically be represented as one of the 2^n values in the root codebook assuming that the sensor uses an n-bit A/D converter. Instead of transmitting n-bits to represent the sensor reading, as would be traditionally done, we can transmit $i < n$ bits if there is side-information,

Y, that is no further than $2^{i-1}\Delta$ away from X available at the decoder. The encoder need only transmit the i bits that specify the subcodebook that X belongs to at level-i, and the decoder will decode Y to the closest value in the subcodebook that the encoder specified. Because Y is no further than $2^{i-1}\Delta$ from the representation of X, the decoder will always decode Y to X. Below, we describe the functionality of the encoder and decoder in detail.

1) Encoder: The encoder will receive a request from the data gathering node requesting that it encode its readings using i bits. The first thing that the encoder does is find the closest representation of the data from the 2^n values in the root codebook (this is typically done by the A/D converter). Next, the encoder determines the subcodebook that X belongs to at level-i. The path through the tree to this subcodebook will specify the bits that are transferred to the data gathering node. The mapping from X to the bits that specify the subcodebook at level i can be done through the following deterministic mapping:

$$f(X) = index(X) \ mod \ 2^i \tag{2}$$

where $f(X)$ represents the bits to be transmitted to the decoder and $index()$ is a mapping from values in the root codebook to their respective indices. For a given X and i, $f(X)$ will be an i-bit value which the data gathering node will use to traverse the tree.

2) Decoder: The decoder (at the data gathering node) will receive the i-bit value, $f(X)$, from the encoder and will traverse the tree starting with the least-significant-bit (LSB) of $f(X)$ to determine the appropriate subcodebook, $\mathcal{S}$ to use. The decoder will then decode the side-information, Y, to the closest value in $\mathcal{S}$:

$$\hat{X} = argmin_{r_i \in \mathcal{S}} ||Y - r_i|| \tag{3}$$

where r_i represents the i^{th} codeword in $\mathcal{S}$. Assuming that Y is less than $2^{i-1}\Delta$ away from X, where Δ is the spacing in the root codebook, then the decoder will be able to decode Y to the exact value of X, and recover X perfectly. The following example will elucidate the encoding/decoding operations.

Example 2: Consider the 4-level tree codebook of Fig. 4. Assume that the data is represented by the value $r_9 = 0.9$ in the root codebook and the data gathering node asks the sensor to encode X using 2 bits. The index of r_9 is 9, so $f(X) = 9 \ mod \ 4 = 1$. Thus, the encoder will send the two bits, 01, to the data gathering node (see Fig. 4). The data gathering node will receive 01 and descend the tree using the least-significant bit first (i.e., 1 and then 0) to determine the subcodebook to decode the side-information with. In the example, we assume that the side-information, Y, is 0.8, and we will decode Y in the subcodebook located at $1, 0$ in the tree to find the closest codeword. This codeword is r_9 which is exactly the value representing X. Thus, we have used 2 bits to convey the value of X instead of using the 4 bits that would have been needed if we had not done any encoding.

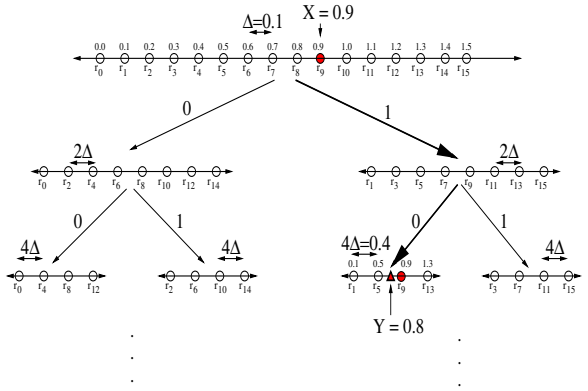

Fig. 4. An example for the tree based codebook. The encoder is asked to encode X using 2 bits, so it transmits 01 to the decoder. The decoder will use the bits 01 in ascending order from the LSB to determine the path to the subcodebook to use to decode Y with.

III. Correlation Tracking

In the above encoding/decoding operations we assume that the decoder for sensor j has available to it at time k some side-information $Y_k^{(j)}$ that is correlated to the sensor reading, $X_k^{(j)}$. In practice, we choose to use a linear predictive model where $Y_k^{(j)}$ is a linear combination of values that are available at the decoder:

$$Y_k^{(j)} = \sum_{l=1}^{M} \alpha_l X_{k-l}^{(j)} + \sum_{i=1}^{j-1} \beta_i X_k^{(i)} \tag{4}$$

where $X_{k-l}^{(j)}$ represents past readings for sensor j and $X_k^{(i)}$ represents present sensor readings from sensor i [2]. The variables α_l and β_i are weighting coefficients. We can then think of $Y_k^{(j)}$ as a linear prediction of $X_k^{(j)}$ based on past values (i.e., $X_{k-l}^{(j)}$; $l = 1, ..., M$) and other sensor readings that have already been decoded at the data gathering node (i.e., $X_k^{(i)}$; $i = 1, ..., j-1$, where i indexes the sensor and $j-1$ represents the number of readings from other sensors that have already been decoded). We choose to use a linear predictive model because it is not only analytically tractable but also optimal in the limiting case where the readings can be modeled as $i.i.d.$ Gaussian random variables.

In order to leverage the inter-node correlations, we require that one of the sensors always sends its data either uncoded or compressed with respect to its own past data. Furthermore, we number the sensors in the order that they are queried. For example, at each time instant, one of the sensors will send its reading, $X_k^{(1)}$, either uncoded or coded with respect to its own past. The reading for sensor 2 can then be decoded with respect to

$$Y_k^{(2)} = \sum_{l=1}^{M} \alpha_l X_{k-l}^{(2)} + \beta_1 X_k^{(1)} \tag{5}$$

[2]Note that for simplicity, our above prediction model is based on a finite number of past values and a single present value for each of the other sensor readings that have been decoded. This model can be generalized to the case where past values of other sensors are also included in the prediction.

Each $X_k^{(i)}$ that is decoded can then be used to form predictions for other sensor readings according to (4). The prediction, $Y_k^{(j)}$, determines the number of bits needed to represent $X_k^{(j)}$. In the extreme case that $Y_k^{(j)}$ perfectly predicts $X_k^{(j)}$ (i.e., $Y_k^{(j)} = X_k^{(j)}$), then zero bits are needed to represent $X_k^{(j)}$ because it is perfectly predictable at the decoder. Thus, the main objective of the decoder is to derive a good estimate of $X_k^{(j)}$ for sensor j, $j = 1, ..., L$, where L represents the number of sensors. In more quantitative terms, we would like for the decoder to be able to find the α_l; $l = 1, ..., M$ and β_i; $i = 1, ..., j-1$ that minimize the mean squared error between $Y_k^{(j)}$ and $X_k^{(j)}$.

To find the α_l and β_i that minimize the mean squared prediction error, let us start by representing the prediction error as a random variable, $N_j = Y_k^{(j)} - X_k^{(j)}$. We can then rewrite the mean squared error as:

$$\begin{array}{c} E[N_j^2] = E[(X_k^{(j)} - \left(\sum_{l=1}^{M} \alpha_l X_{k-l}^{(j)} + \sum_{i=1}^{j-1} \beta_i X_k^{(i)}\right))^2] \\ = E[X_k^{(j)2}] - 2\sum_{l=1}^{M} \alpha_l E[X_k^{(j)} X_{k-l}^{(j)}] - \\ 2\sum_{i=1}^{N} \beta_i E[X_k^{(j)} X_k^{(i)}] + 2\sum_{l=1}^{M}\sum_{i=1}^{j-1} \alpha_l \beta_i E[X_{k-l}^{(j)} X_k^{(i)}] \\ + \sum_{l,h=1}^{M} \alpha_l \alpha_h E[X_{k-l}^{(j)} X_{k-h}^{(j)}] + \sum_{i,h=1}^{j-1} \beta_i \beta_h E[X_k^{(i)} X_k^{(h)}] \end{array}$$

Now, if we assume that $X_k^{(j)}$ and $X_k^{(i)}$ are pairwise jointly wide sense stationary [10] for $i = 1, ..., j-1$, then we can re-write the mean squared error as:

$$E[N_j^2] = r_{x^j x^j}(0) - 2\vec{P}_j^T \vec{\Gamma}_j + \vec{\Gamma}_j^T R_{zz}^j \vec{\Gamma}_j \tag{6}$$

where

$$\vec{\Gamma}_j = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ ... \\ \alpha_M \\ \beta_1 \\ \beta_2 \\ ... \\ \beta_{j-1} \end{bmatrix}, \quad \vec{P}_j = \begin{bmatrix} r_{x^j x^j}(1) \\ r_{x^j x^j}(2) \\ ... \\ r_{x^j x^j}(M) \\ r_{x^j x^1}(0) \\ r_{x^j x^2}(0) \\ ... \\ r_{x^j x^{j-1}}(0) \end{bmatrix}$$

and we use the notation $r_{x^j x^i}(l) = E[X_k^j X_{k+l}^i]$. With this notation, we can express R_{zz}^j as:

$$R_{zz}^j = \begin{bmatrix} R_{x^j x^j} & R_{x^j x^i} \\ R_{x^j x^i}^T & R_{x^i x^i} \end{bmatrix}$$

where $R_{x^j x^j}$ is given as:

$$\begin{bmatrix} r_{x^j x^j}(0) & r_{x^j x^j}(1) & . & r_{x^j x^j}(M-1) \\ r_{x^j x^j}(1) & r_{x^j x^j}(0) & . & r_{x^j x^j}(M-2) \\ ... & ... & . & ... \\ r_{x^j x^j}(M-1) & r_{x^j x^j}(M-2) & . & r_{x^j x^j}(0) \end{bmatrix},$$

and $R_{x^j x^i}$ and $R_{x^i x^i}$ are given as:

$$R_{x^j x^i} = \begin{bmatrix} r_{x^j x^1}(1) & r_{x^j x^2}(1) & ... & r_{x^j x^{j-1}}(1) \\ r_{x^j x^1}(2) & r_{x^j x^2}(2) & ... & r_{x^j x^{j-1}}(2) \\ ... & ... & ... & ... \\ r_{x^j x^1}(M) & r_{x^j x^2}(M) & ... & r_{x^j x^{j-1}}(M) \end{bmatrix}$$

and

$$R_{x^i x^i} = \begin{bmatrix} r_{x^1 x^1}(0) & r_{x^1 x^2}(0) & ... & r_{x^1 x^{j-1}}(0) \\ r_{x^2 x^1}(0) & r_{x^2 x^2}(0) & ... & r_{x^2 x^{j-1}}(0) \\ ... & ... & ... & ... \\ r_{x^{j-1} x^1}(0) & r_{x^{j-1} x^2}(0) & ... & r_{x^{j-1} x^{j-1}}(0) \end{bmatrix}$$

To find the set of coefficients (represented by $\vec{\Gamma}_j$) that minimize the mean squared error, we differentiate (6) with respect to $\vec{\Gamma}_j$ to obtain:

$$\frac{\partial E[N_j^2]}{\partial \vec{\Gamma}_j} = -2\vec{P}_j + 2R_{zz}^j \vec{\Gamma}_j \tag{7}$$

Setting the above equal to zero and solving for the optimal $\vec{\Gamma}_j$, which we denote by $\vec{\Gamma}_{j,opt}$, we arrive at the standard Wiener estimate [10]:

$$\vec{\Gamma}_{j,opt} = R_{zz}^{-1,j} \vec{P}_j \tag{8}$$

If our assumption of stationarity holds, then the data gathering node can request for uncoded data from all of the sensors for the first K rounds of requests and calculate the Wiener estimate (8) once from these K rounds of samples. The set of coefficients determined from the Wiener estimate can then be used to form the side information for each future round of request. In practice, however, the statistics of the data may be time varying and as a result, the coefficient vector, $\vec{\Gamma}_j$, must be continuously adjusted to minimize the mean-squared error. One method of doing this is to move $\vec{\Gamma}_j$ in the opposite direction of the gradient of the objective function (i.e., the mean squared error) for each new sample received during round $k+1$:

$$\vec{\Gamma}_j^{(k+1)} = \vec{\Gamma}_j^{(k)} - \mu \nabla_j^{(k)} \tag{9}$$

where $\nabla_j^{(k)}$ is given by (7) and μ represents the amount to descend opposite to the gradient. The goal of this approach is to descend to the global minima of the objective function. We are assured that such a minima exists because the objective function is convex. In fact, it has been shown, that if μ is chosen correctly then (9) will converge to the optimal solution [10]. In the following subsection we will show how (9) can be calculated in practice and how to incorporate adaptive prediction with the distributed source code discussed in the previous section.

A. Parameter estimation

From (7) and (9), we know that the coefficient vector should be updated as:

$$\vec{\Gamma}_j^{(k+1)} = \vec{\Gamma}_j^{(k)} - \frac{1}{2}\mu(-2\vec{P}_j + 2R_{zz}^j \vec{\Gamma}_j^{(k)}). \tag{10}$$

In practice, however, the data gathering node will not have knowledge of $\vec{P}_j$ and R_{zz}^j and will therefore need an efficient method for estimating $\vec{P}_j$ and R_{zz}^j. One standard estimate is

to use $\vec{P}_j = X_k^{(j)} \vec{Z}_{k,j}$ and $R_{zz} = \vec{Z}_{k,j} \vec{Z}_{k,j}^T$ where

$$\vec{Z_{k,j}} = \begin{bmatrix} X_{k-1}^{(j)} \\ X_{k-2}^{(j)} \\ \cdots \\ X_{k-M}^{(j)} \\ X_k^{(1)} \\ X_k^{(2)} \\ \cdots \\ X_k^{(j-1)} \end{bmatrix}$$

so that (10) becomes

$$\begin{aligned} \vec{\Gamma}_j^{(k+1)} &= \vec{\Gamma}_j^{(k)} - \mu \vec{Z}_{k,j}(-X_k^{(j)} + \vec{Z}_{k,j}^T \vec{\Gamma}_j^{(k)}) \\ &= \vec{\Gamma}_j^{(k)} + \mu \vec{Z}_{k,j} N_{k,j} \end{aligned} \quad (11)$$

where the second equality follows from the fact that $Y_k^{(j)} = \vec{Z}_{k,j}^T \vec{\Gamma}_j^{(k)}$ and $N_{k,j} = X_k^{(j)} - Y_k^{(j)}$. The equation described by (12) is well known in the adaptive filtering literature as the Least-Mean-Squares (LMS) algorithm and the steps in calculating the LMS solution is summarized below:

1. $Y_k^{(j)} = \vec{\Gamma}_j^{(k)T} \vec{Z_{k,j}}$
2. $N_{k,j} = X_k^{(j)} - Y_k^{(j)}$
3. $\vec{\Gamma}_j^{(k+1)} = \vec{\Gamma}_j^{(k)} + \mu \vec{Z}_{k,j} N_{k,j}$

To use the LMS algorithm, the data gathering node will start by querying all of the sensors for uncoded data for the first K rounds of requests. The value of K should be chosen to be large enough to allow the LMS algorithm to converge. After K rounds of requests have been completed, the data gathering node can then ask for coded values from the sensor nodes and decode the coded value for sensor j with respect to its corresponding side information, $Y_k^{(j)} = \vec{\Gamma}_j^T \vec{Z}_{k,j}^{(j)}$. The value of $\vec{\Gamma}_j$ will continue to be updated to adjust to changes in the statistics of the data. More specifically, for each round of request and each value reported by a sensor, the decoder will decode $Y_k^{(j)}$ to the closest codeword in the subcodebook, $\mathcal{S}$, specified by the corresponding sensor

$$\hat{X_k}^{(j)} = argmin_{r_i \in \mathcal{S}} ||Y_k^{(j)} - r_i|| \quad (12)$$

From section II-B, we know that $\hat{X_k}^{(j)}$ will always equal $X_k^{(j)}$ as long as the sensor node encodes $X_k^{(j)}$ using i bits so that $2^{i-1}\Delta > |N_{k,j}|$. If $|N_{k,j}| > 2^{i-1}\Delta$, however, then a decoding error will occur. We can use Chebyshev's inequality [11] to bound this probability of error:

$$P[|N_{k,j}| > 2^{i-1}\Delta] \leq \frac{\sigma_{N_j}^2}{(2^{i-1}\Delta)^2} \quad (13)$$

where $N_{k,j}$ is drawn from a distribution with zero mean and variance $\sigma_{N_j}^2$. Thus, to insure that $P[|N_{k,j}| > 2^{i-1}\Delta]$ is less than some probability of error, P_e, we can choose $\frac{\sigma_{N_j}^2}{(2^{i-1}\Delta)^2} = P_e$. The value of i that will insure this probability of error is then given as

$$i = \frac{1}{2} log_2(\frac{\sigma_{N_j}^2}{\Delta^2 P_e}) + 1 \quad (14)$$

Thus, for a given P_e, the data gathering node should ask for i-bits from each sensor according to (14). Note that it is not necessary to be over-conservative when choosing P_e because Chebyshev's inequality is a loose bound.

From (14), we can see that the data gathering node must maintain an estimate of the variance of the prediction error, $\sigma_{N_j}^2$, for each sensor in order to determine the number of bits to request from each sensor. The data gathering node can initialize $\sigma_{N_j}^2$ as:

$$\sigma_{N_j}^2 = \frac{1}{K-1} \sum_{i=1}^{K} N_{k,j}^2 \quad (15)$$

during the first K rounds of requests. To update $\sigma_{N_j}^2$, the data gathering node can form the following filtered estimate:

$$\sigma_{N_j,new}^2 = (1-\gamma)\sigma_{N_j,old}^2 + \gamma N_{k,j}^2 \quad (16)$$

where $\sigma_{N_j,old}^2$ is the previous estimate of $\sigma_{N_j}^2$ and γ is a "forgetting factor" [10]. We choose to use a filtered estimate to adapt to changes in statistics.

B. Decoding error

As mentioned above, it is always possible for the data gathering node to make a decoding error if the magnitude of the correlation noise, $|N_{k,j}|$, is larger than $2^{i-1}\Delta$ where i is the number of bits used to encode the sensor reading for sensor j at time k. We propose two approaches for dealing with such errors. One method is to use error detection codes and the other method entails using error correction codes.

To use error detection, each sensor node can transmit a cyclic redundancy check (CRC) [12] for every m readings that it transmits. The data gathering node will decode the m readings using the tree-structured codebook as above and compare its own calculation of the CRC (based on the m readings it decodes) to the CRC transmitted by the sensor. If an error is detected (i.e, the CRC does not match), then the data gathering node can either drop the m readings or ask for a retransmission of the m readings. Whether the data gathering node drops the m readings or asks for a retransmission is application dependent, and we do not address this issue in this paper. Furthermore, by using Chebyshev's inequality (13), the data gathering node can make the probability of decoding error as small as it desires which translates directly into a lower probability of data drops or retransmissions.

The other method of guarding against decoding error is to use error-correction codes. We propose using a non-binary error correction code such as an (M,K) Reed-Solomon code [13] that can operate on K sensor readings and generate $M-K$ parity check symbols. These $M-K$ parity check symbols can be transmitted to the data gathering node along with the K encoded sensor readings. The data gathering node will decode the K sensor readings using the tree-based structure mentioned above and upon receiving the $M-K$ parity check symbols, it can correct for any errors that occurred in the K sensor readings. If more than $\frac{M-K}{2}$ errors exist in the K sensor readings, then the Reed-Solomon decoder will

declare that the errors can not be corrected and in this case, the data must be either dropped or retransmitted.

IV. Querying and reporting algorithm

In this section, we combine the concepts of the previous sections to formulate the algorithms to be used by the data gathering node and by the sensor node.

A. Data gathering node algorithm

The data gathering node will, in general make N rounds of queries to the sensor nodes. In the first K rounds of queries, the data gathering node will ask the sensors to send their data uncoded. The reason for this is that the data gathering node needs to determine the correlation structure between sensor readings before asking for compressed readings. Thus, the data gathering node will use the first K rounds of readings for calculating the correlation structure in accordance with Sec. III. After K rounds of readings, the data gathering node will have an estimate of the prediction coefficients to be used for each sensor (see (10)). Note that K should be chosen large enough to allow for the LMS algorithm to converge. For each round after K, one node will be asked to send its reading "uncompressed" with respect to the other sensors [3]. The data gathering node will alternate the requests for "uncompressed" data among the nodes to insure that no single node is asked to expend more energy than the others. Upon receiving a transmission from a sensor, the data gathering node will decode it (if it is a compressed reading) with respect to a linear estimate of the data for that sensor (see (4)). After each round of requests, the correlation parameters of each sensor (see (10) and (16)) are updated. Pseudocode for the data gathering node is given below.

Pseudocode for data gathering node:

Initialization:
for $(i = 0; i < K; i++)$
  for $(j = 0; j < num\_sensors; j++)$
    Ask sensor j for its uncoded reading
  for each pair of sensors i,j
    update correlation parameters using Eqs. (16) and (10).

Main Loop:
for $(k = K; k < N; k++)$
  Request a sensor for uncoded reading
  for each remaining sensor
    determine number of bits, i, to request for using Eq.(14).
    Request for i bits
  Decode data for each sensor.
  Update correlation parameters for each sensor.

[3]Note that the sensor may still send its data compressed with respect to its own past

The decoding is done in accordance with Sec. II-B and the correlation parameters are estimated according to Eq. (10) and (16).

B. Sensor node algorithm

The algorithm incorporated into each sensor node is considerably simpler than the algorithm incorporated into the data gathering node. The sensor node will simply listen for requests from the data gathering node. The data gathering node will specify to the sensor the number of bits that it requests the sensor to encode the data with. Each sensor will be equipped with an A/D converter that represents the data using n-bits. Upon receiving a request from the data gathering node, the sensor will encode the n-bit value from the A/D converter using i-bits, where i is specified by the data gathering node. This i-bit value is sent back to the data gathering node. Pseudocode for the sensor node is given below.

Pseudocode for sensor nodes:

For each request
  Extract i from the request
  Get X[n] from A/D converter
  Transmit n mod 2^i

In the above algorithm, we denote $X[n]$ as the value returned from the A/D converter and n as the index to this value. Note that the only extra operation with respect to an uncoded system is for the sensor nodes to perform a modulo operation. This makes it extremely cheap for a sensor node to encode its data.

V. Simulation results

In this section we provide simulation results. The simulations were performed for measurements on light, temperature and humidity. We wanted to measure not only the energy savings (due to bit transmissions) but also our correlation tracking algorithm and the robustness of our algorithm to errors. We implemented both the data gathering node algorithm and the sensor node algorithm described in Sec. IV. In our first set of simulations, we ran the data gathering algorithm on one machine and the sensor node algorithms on a set of different machines. The sensor node machines simulated the measurement of data by reading from a file, previously recorded readings from actual sensors. The data measured by the sensors were for light, humidity and temperature. We assumed a 12 bit A/D converter with a dynamic range of $[-128, 128]$ in our simulations and further assumed a star topology where the data gathering node queried 5 sensor nodes directly.

A. Correlation tracking

The first simulation that we ran was to test our correlation tracking algorithm (see Sec. III). We modeled the data received

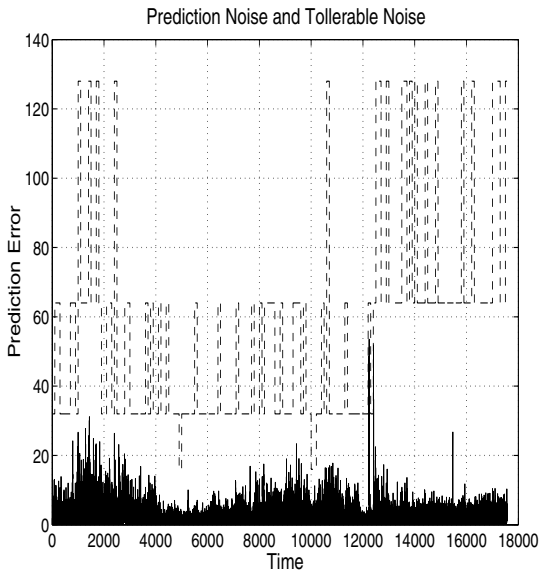

Fig. 5. Tolerable noise vs. prediction noise for 18,000 samples of humidity. The tolerable noise is the amount of noise that can exist between the prediction of a sensor reading and the actual sensor reading without inducing a decoding error.

by sensor j as:

$$Y_k^{(j)} = \sum_{l=1}^{4} \alpha_l X_{k-l}^{(j)} + X_k^{(m)} \tag{17}$$

where $m \neq j$. In other words the prediction of the reading for sensor j is derived from its own past values and one other sensor. To test the correlation tracking algorithm, we measured the tolerable noise that the correlation tracking algorithm calculates at each time instant. The tolerable noise is the amount of noise that can exist between the prediction of a sensor reading and the actual sensor reading without inducing a decoding error. Tolerable noise is calculated by using (14), and noting that the tolerable noise will be given as $2^{i-1}\Delta$ where i is the number of bits that are requested from the sensor and Δ is the spacing of values in the A/D converter. We set the bound on probability of decoding error to be less than 1 in 100 and simulated the data gathering algorithm and the sensor node algorithms over 18,000 samples of light, temperature and humidity for each sensor (a total of 90,000 samples). A plot of the tolerable noise vs. actual prediction noise is given in Fig. 5. where the top graph represents the tolerable noise and the bottom graph represents the actual prediction noise.

From the plot it can be seen that the tolerable noise is much larger than the actual prediction noise. The reason for this is that we were conservative in choosing the parameters for estimating the number of bits to request from the sensors. The tolerable noise can be lowered to achieve higher efficiency but this also leads to a higher probability of decoding error. For the simulations that we ran, zero decoding errors were made for 90,000 samples of humidity, temperature and light.

One other thing to note from the plot is that there are many spikes in the tolerable noise. These spikes occur because we chose an aggressive weighting factor for calculating (16). These spikes can be reduced by weighting the current distortion less in the estimation of the overall distortion (see (16)), but this will lead to slower responses to variations in distortion and will therefore introduce more decoding errors for noisy data.

B. Energy savings

The next set of simulations were run to measure the amount of energy savings that the sensor nodes achieved. The energy savings were calculated to be the total reduction in energy that resulted from transmission and reception. Note that for reception, energy expenditure is actually not reduced but increased because the sensor nodes need to receive the extra bits that specify the number of bits to encode each sensor reading with. For an n-bit A/D converter, an extra $log(n)$ bits need to be received each time the data gathering node informs a sensor of the number of bits needed for encoding. We assume that the energy used to transmit a bit is equivalent to the energy used to receive a bit. To reduce the extra energy needed for reception, we simulated the data gathering node to only specify the number of encoding bits periodically. In our simulations, we chose for this period to be 100 samples for each sensor node. The 5 sensor nodes were alternately queried to send back readings that were compressed only with respect to its own past readings so that compressed readings from other sensors could be decoded with respect to these readings. The overall average savings in energy is given in Table 1. To assess the performance of our algorithm, we choose to use the work of [14] as a benchmark for comparison. The work of [14] is also based on a distributed coding framework but the prediction algorithm uses a filtered estimate for the prediction coefficients instead of using a gradient descent algorithm such as LMS to determine the prediction coefficients. Furthermore, in [14] the prediction algorithm only uses one measurement from a neighboring sensor to form the prediction estimate. Thus, in order to perform a fair comparison, we changed the model of (17) to only use one measurement from another sensor to form the prediction estimate and surprisingly was able to achieve roughly the same performance as given in Table 1. The results for humidity are approximately 24% better than the results cited in [14] for the same data set. Similarly, the results for temperature and light are approximately 16% and 3% better respectively than the results cited in [14] for the respective data sets. Thus, it is clear that the LMS algorithm is better suited for tracking correlations than the methods given in [14].

Data Set	Temperature	Humidity	Light
Ave Energy Savings	66.6%	44.9%	11.7%

TABLE I

AVERAGE ENERGY SAVINGS OVER AN UNCODED SYSTEM FOR SENSOR NODES MEASURING TEMPERATURE, HUMIDITY AND LIGHT

One can achieve even larger energy savings than the savings cited above by using a less conservative estimate of the bits

needed for encoding (see (Eq. 14)). This will, however, lead to more decoding errors. In our simulations we chose a bound on the probability of decoding error that resulted in 0 decoding errors over 90,000 samples for each of the data sets. In the following subsection, we will evaluate the robustness of our algorithm to errors.

C. Robustness to errors

There are two types of errors that can occur in our framework. The first type of error is a packet loss. The second type of error is an actual decoding error which results from the code not being able to correct for the prediction noise. We will consider each type of error in the following subsections.

1) Packet loss: A packet loss may occur if a measurement is lost due to a malfunction of the sensor or if there is a transmission loss. In such a case, it appears that this loss should affect the prediction estimates that depend on this measurement (see (17)). This is not true, however, because the prediction algorithm may replace this measurement with a previous measurement from the same sensor to form the prediction estimate. In fact, we tested a few scenarios in which the packet drop rate was approximately 10% and we were able to achieve the same compression rate with zero decoding errors. Thus, our algorithm has the *additional feature that it is robust to packet loss*.

2) Decoding error: The other type of error is a decoding error. Recall, in Sec. III-B, we mentioned that it is possible to make a decoding error if the actual prediction noise between the prediction estimate and the sensor reading exceeds the tolerable noise specified by the data gathering node. One can bound this probability of decoding error by using Chebyshev's inequality to specify the number of bits needed for encoding (see (14)). But Chebyshev's inequality is a loose bound, as can be seen from Fig. 5 and as a result, it is difficult to determine the minimal number of bits that need to be sent by each sensor without inducing a decoding error. We can therefore see that there is a delicate trade-off between energy savings and decoding error.

To achieve both large energy savings and robustness, the data gathering node can use an aggressive estimate of the number of bits that is needed from each sensor and each sensor can apply an error detection code or error correction code to its readings so that the data gathering node can handle decoding errors appropriately. The other alternative is for the data gathering node to over-estimate the number of bits needed for encoding to decrease the decoding error. This is the approach we took in our simulations (we chose a bound such that the decoding error was 0), but the downside to this approach is that there is a corresponding decrease in energy savings for the sensor nodes.

VI. Conclusion

We have proposed a method of reducing energy consumption in sensor networks by using distributed compression and adaptive prediction. Distributed compression leverages the fact that there exist inherent correlations between sensor readings and hence sensor readings can be compressed with respect to past sensor readings and sensor readings measured by other nodes. We proposed a novel method for allowing nodes to compress their readings to different levels without having the nodes know what the other nodes are measuring. Adaptive prediction is used to track the correlation structure of the sensor network, and ultimately determined the number of bits needed to be spent by the sensor nodes. This approach appears to be promising, as preliminary results show that an average energy savings per sensor node of $10 - 65\%$ can be achieved using our algorithm.

The energy savings achieved through our simulations are a conservative estimate of what can be achieved in practice. In practice, one can use richer models at the data gathering node to describe the correlation structure in the sensor network. We chose to use a simple predictive model in our simulations to demonstrate the power of our approach. In addition, our algorithm can be combined with other energy-saving approaches such as data aggregation to achieve additional gains. Future work remains in exploring more robust codes for the sensor nodes and better predictive models for the data gathering node along with incorporating our algorithm with energy-saving routing algorithms.

Acknowledgment

The authors would like to thank Ion Stoica, Rahul Shah and Jan Rabaey for some stimulating conversations.

References

[1] C. Toh, "Maximum battery life routing to support ubiquitous mobile computing in wireless ad hoc networks," *IEEE Communications Magazine*, pp. 138–147, June 2001.

[2] R. Shah and J. Rabaey, "Energy aware routing for low energy ad hoc sensor networks," *Proc. of IEEE WCNC*, Mar 2002.

[3] D. E. C. Intanagonwiwat, R. Govindan, "Directed diffusion: A scalable and robust communication paradigm for sensor networks," *Proc. of IEEE MobiCom*, Aug 2000.

[4] G. Pottie and W. Kaiser, "Wireless sensor networks," *Communications of the ACM*, 2000.

[5] M. Chu, H. Haussecker, and F. Zhao, "Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks," *IEEE Journal of High Performance Computing Applications*, To Appear 2002.

[6] D. Slepian and J. K. Wolf, "Noiseless encoding of correlated information sources," *IEEE Trans. on Inform. Theory*, vol. IT-19, pp. 471–480, July 1973.

[7] T. M. Cover and J. A. Thomas, *Elements of Information theory*. New York: Wiley, 1991.

[8] A. D. Wyner and J. Ziv, "The rate-distortion function for source coding with side information at the decoder," *IEEE Trans. on Inform. Theory*, vol. IT-22, pp. 1–10, January 1976.

[9] S. S. Pradhan and K. Ramchandran, "Distributed source coding using syndromes: Design and construction," *Proceedings of the Data Comression Conference (DCC)*, March 1999.

[10] S. Haykin, *Adaptive Filter Theory*. Upper Saddle River: Prentice Hall, 1996.

[11] H. Stark and J. Woods, *Probability, Random Processes and Estimation Theory for Engineers*. Englewood Cliffs: Prentice Hall, 1994.

[12] T. Ramabadran and S. Gaitonde, "A tutorial on crc computations," *IEEE Micro*, vol. 45, pp. 62–74, Aug 1988.

[13] R. Blahut, *Theory and Practice of Data Transmission Codes*, 1995.

[14] J. Chou, D. Petrovic, and K. Ramchandran, "Tracking and exploiting correlations in dense sensor networks," *Proceedings of the Asilomar Conference on Signals, Systems and Computers*, November 2002.