9195

DISTRIBUTED SIGNAL COMPRESSION AND AGGREGATION FOR SENSOR NETWORKS

An-swol Hu *

Cornell University
School of Electrical and Computer Engineering
ach25@cornell.edu

Prasanta Bose

Lockheed Martin Space Systems Company
Advanced Technology Center
prasanta.bose@lmco.com

ABSTRACT

We present the Distributed Signal Compression and Aggregation (DSCA) method to solve the signal exfiltration problem. The signal exfiltration problem is the task of taking distributed observations of a particular signal and exfiltrating that information to some base station node that needs the signal information. We develop a model for distributed signal sampling and develop a wavelet transform based compression method to reduce the size of the data set that needs to be communicated. A method for averaging together signal information is devised to achieve signal aggregation. MATLAB and Ptolemy II simulation results are presented to analyze the performance of the DSCA method.

1. INTRODUCTION

In recent years, advancements in embedded technology have allowed for the possibility of employing distributed sensor networks for a variety of monitoring and surveillance tasks. Sensor networks have the ability to give us unprecedented capabilities in the observation and control of the physical world and these networks have applications in everything from battlefield surveillance to habitat monitoring [7].

In applying large-scale sensor networks to such monitoring applications, we are inherently confronted with the challenge of extracting the sensed information from the distributed sensor nodes. One particular problem that appears in many applications is the need to extract a time-varying signal being produced by some target source. The extracted signal may then be used, for example, in target classification. This problem is known as the *signal exfiltration problem* illustrated in Fig. 1. For example, in [3] the authors consider the problem of classifying birdcalls. They devise a preprocessing method for a tiered sensor network to detect and classify birdcalls. However, the desire to move the detected birdcall or extract an unclassified birdcall is clearly the signal exfiltration problem. Another significant application is found in the use of unattended ground sensor systems (UGS) [6],[5] for target tracking and classification. In these networks, ground sensors seek to classify or track targets based on the seismic vibration patterns produced by different types of targets. In this case, the problem can be solved by moving the detected waveforms to a more powerful sensor node for classification. This idea can also be applied to the use of acoustic sensors [4] for target classification.

The signal exfiltration problem presents us with two primary challenges. First, nearly all sensor networks are severely power constrained and thus there is a need to conserve power. Since wireless communication puts a significant demand on the power resources of each node, we would like to compress the signal and reduce the communication requirements. Second, since many nodes may observe the same target signal, we would like to combine these distributed observations to improve the final signal communicated to the "base station" node that wants to extract the signal. As a result, we can also describe the signal exfiltration problem as a distributed compression and aggregation problem. Another powerful way to consider the signal exfiltration problem is to view it as a distributed signal sampling and aggregation problem. Many different nodes will sample the target signal and the goals is to combine all the samples in a distributed manner so the base station node will have the best possible sample set.

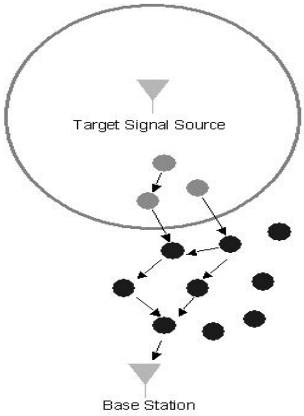

Fig. 1. The signal exfiltration problem. A target signal source is detected by a network and the light gray nodes make observations of the signal. The signal information is moved through the network to a base station node.

The remainder of the paper is organized as follows. In Section 2 we setup the problem and describe the sampling observations made by each node. We describe our Distributed Signal Compression and Aggregation (DSCA) method in Section 3 and present simulation and analysis results in Section 4.

2. PROBLEM SETUP

2.1. System Overview

We consider N nodes densely deployed over a finite area along with a stationary signal source emitting a bandlimited signal of unknown bandwidth no greater than B. The signal source either lies within the area of the network or close enough to the network so that N_R nodes, where $0 < N_R \leq N$, are within distance R of the source and can observe the transmitted signal. We call R the transmission range of the signal source and we assume that all nodes within the transmission range of the source will hear the same time-varying signal while all nodes outside the transmission range will hear nothing. These assumptions are made since we assume an RF signal source which results in negligible phase shift and amplitude loss over

*This work was done while the author was a summer intern with Lockheed Martin Space Systems Company, Advanced Technology Center.

a small transmission distance R. For example, since wireless signals suffer a time delay of approximately 1 ns/foot [1], a low frequency signal will have negligible phase shift over a distance of a few feet.

A *base station* node is placed in the sensor network. This base station node wants to obtain a finely sampled version of the signal transmitted by the signal source sampled at K Hz, where $K > 2B$. This node, however, may or may not be in the transmission range of the signal source, so for generality we assume the base station node can not directly sample the signal and must receive its information from surrounding sensor nodes. Thus, the task of the senor network has two primary parts. First, the nodes in the network that are within the transmission range of the signal source must make samples of the source signal. Second, the network must aggregate the signal samples made my these nodes and move this information to the base station node.

The operation of the network will therefore be divided into two phases: a *signal sampling* phase and a *signal processing* phase. The network will continually alternate between signal sampling and signal processing. During a signal sampling phase, each node in the transmission range of the source signal will make noisy samples of the transmitted signal and obtain a low quality reconstruction of the source signal (the details of this phase will be described in Section 2.2). In a signal processing phase each node will execute the DSCA (*D*istributed *S*ignal *C*ompression and *A*ggregation) method and move the signal information obtained during the previous sampling phase to the base station node. The method is described in Section 3 and we assume existing data link and networking layers that handle node to node communication to move information to the base station node. We further assume the existence of a synchronization protocol so all nodes in the network are closely synchronized. That is, all nodes enter and leave a particular phase, either signal sampling or signal processing, at the same time.

2.2. Signal Sampling

Even though we assume the nodes are synchronized, we do not assume that the nodes take samples at the same time. As well, we make the average sampling rate of each node much less than the K Hz desired by the base station node. The reasoning for this is that during a sampling phase it is desirable to allow nodes to sample whenever it is convenient for it to sample and reduce the sampling requirements for any one node. This will reduce computational complexity and power requirements. To get the desired K Hz sampling rate, we will seek to combine the information from all nodes that make samples and reconstruct the signal from the aggregated data.

To model this sampling behavior, for any given sampling phase we first define a *possible sampling point* as an instant in time that the base station node would have liked to sample. Thus, a possible sampling point occurs every $1/K$ seconds. At each possible sampling point, the node samples with probability p and does not sample with probability $1 - p$, independent of all other possible sample points or other nodes. If a node samples at a particular possible sample point, that sample point becomes an *actual sample point*. At each actual sampling point, the source signal value is corrupted by independent additive Gaussian noise with mean 0 and variance σ^2. It is important to note that the random sampling at each possible sampling point is simply a model for generating the sample set. The manner in which each node takes samples is unimportant as long as each node samples on a subset of all possible sampling points.

After a node does this random sampling, we fill in the other possible sampling points at which the node did not sample by using linear interpolation between any two actual sample points. The possible sample points before the first actual sample point and after the last actual sample point are set to the value of the first actual sample point and the last actual sample point, respectively. This gives the node an approximate fine sampling of the noisy source signal.

We choose to use linear interpolation to construct the sampled signal since it is the most straight forward method. This approach does not give perfect reconstruction of the original signal but is considerably less computationally intensive than methods for reconstructing signals from irregular samples [2].

The vector of sampled values and linearly interpolated values is called the *samples vector*. As well, during sampling the node will create a *samples number vector* that is of the same length as the samples vector with each element corresponding to a possible sampling point. The samples number vector will have a 1 at each actual sample point and a 0 at all other possible sampling points. Thus, the samples number vector can keep track of which sample values were actually sampled and which were linearly interpolated. We will thus consider a *data set* to contain a samples vector and a samples number vector.

3. THE DSCA METHOD

The DSCA method comprises of two main parts: signal compression and signal aggregation. We describe each in detail before outlining the entire method.

3.1. Signal Compression

The signal compression that is done in the DSCA method reduces the number of samples needed to represent the samples vector. The affects of quantization of the sample values are not considered in this work. We focus on this lossy compression in an effort to not only reduces the length of the samples vector but also to denoise the signal as well. In actual system implementation, the bit representation of the sample values is another parameter that can be adjusted to suit the particular application. As well, the compression of the samples number vector is not considered here since lossless compression is desired for this vector. A particular lossless compression algorithm should be chosen based on the system application when the method is implemented.

Signal compression is carried out in three main steps. First, an M-level wavelet transform is performed on the samples vector. Second, we set a number of transform values to zero to retain at least `percentEnergyCapture` percent of the signal energy. Third, we discard all but `numWaveletSamples` elements of the samples vector to achieve compression. The steps are now described in detail. We further discuss the effects and trade-offs associated with the parameters in Section 4.

The first step is to do an M-level wavelet transform on the samples vector. If we have a samples vector of length L_{init}, then we first zero-pad the length to $L = L_{init} + L_p = 2^{M_{max}}$, where L_p is the smallest positive integer such that the expression holds for some positive integer M_{max}. We do this step so that M can then be chosen to be $M \leq M_{max}$. In general, this step is not required as the wavelet transform can be done to as many levels as the number of times L_{init} can be divided by 2. The M found without zero-padding, however, will be less than or equal to the M value found after zero-padding. Having the ability to choose a larger M value gives us more flexibility in compressing a variety of signals.

The choice of the particular wavelet transform should also be determined by the particular application. In this work we choose to use the Daubechies-4 wavelet transform [8] since the scaling signals and wavelets have short support thus minimizing computation. The sup-

port is longer than the Haar wavelet transform but the Daubechies-4 transform performs well for a wider variety of signals.

The second step is to retain `percentEnergyCapture` percent of the signal energy. We define the energy E_1^L of a signal $\{a_k\}$, $k = 1, \ldots, L$ as

$$E_1^L = \sum_{k=1}^{L} a_k^2.$$

More generally we define the energy of a sub-signal $\{a_m, \ldots, a_n\}$ as

$$E_m^n = \sum_{k=m}^{n} a_k^2.$$

We thus define an energy profile for the signal $\{a_k\}$, $k = 1, \ldots, L$ as the set of numbers

$$\left\{ \frac{E_1^1}{E_1^L}, \frac{E_1^2}{E_1^L}, \frac{E_1^3}{E_1^L}, \ldots, \frac{E_1^{L-1}}{E_1^L}, 1 \right\}.$$

We show a signal with its corresponding wavelet transform and energy profile in Fig. 2. To capture `percentEnergyCapture` percent of the signal energy, we search the energy profile of the wavelet transform and find the smallest index j such that

$$E_1^j / E_1^L \geq \texttt{percentEnergyCapture}/100.$$

This search starts from the left of the transformed signal (where the M-level average signal resides) to the right (where the 1-level detail signal resides). We then set to zero all values in indices $j + 1$ to L of the wavelet transform.

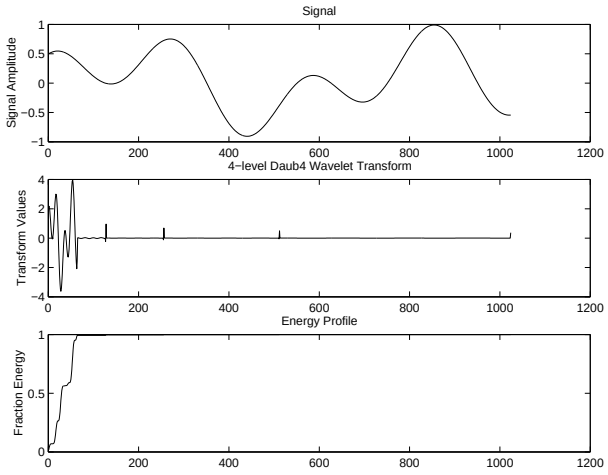

Fig. 2. The above figure illustrates a signal with its corresponding wavelet transform and energy profile. The signal is shown at the top. The wavelet transform (middle) is a 4-level Daubechies-4 wavelet transform of the signal. The energy profile of the wavelet transform is shown at the bottom.

The third step is to retain only `numWaveletSamples` of the wavelet transform. We keep the first `numWaveletSamples` starting from the left and set the rest of the values to zero. This means that the node will only need to send `numWaveletSamples` of data instead of the original L samples.

3.2. Signal Aggregation

The signal aggregation step allows the DSCA method to combine any two data sets to yield a new combined data set. If a node has more than two data sets to combine, then it will start by combining the first two data sets. The new combined data set will then be combined with the third data set. The output of this combination will then be combined with the fourth data set. It will proceed in this manner until there is only one data set that incorporates all the original data sets. Thus, we focus only on the combination of two data sets.

Consider two data sets, data set 1 and data set 2. The procedure for signal aggregation starts by considering each nth element in the two samples number vectors. Assume that the nth element of samples number vector 1 (the samples number vector in data set 1) is an integer c_1 and the nth element of samples number vector 2 (the samples number vector in data set 2) is c_2. Then we have four cases: (1) $c_1 > 0$ and $c_2 > 0$, (2) $c_1 = 0$ and $c_2 = 0$, (3) $c_1 > 0$ and $c_2 = 0$, and (4) $c_1 = 0$ and $c_2 > 0$.

If $c_1 > 0$ and $c_2 > 0$, then we know that the nth element of samples vector 1 and samples vector 2 are sampled values or are derived from sampled values. The nth element of the combined samples vector will be

$$\text{CSV}[n] = \frac{\Big(\text{SV1}[n]\text{SNV1}[n] + \text{SV2}[n]\text{SNV2}[n]\Big)}{\text{SNV1}[n] + \text{SNV2}[n]}$$

and the nth element of the combined samples number vector will be

$$\text{CSNV}[n] = \text{SNV1}[n] + \text{SNV2}[n]$$

where SVi[n] is the nth element of samples vector i, SNVi[n] is the nth element of samples number vector i, CSV[n] is the nth element of the combined samples vector, and CSNV[n] is the nth element of the combined samples number vector, with $i \in \{1, 2\}$.

If $c_1 = 0$ and $c_2 = 0$, then we know that the nth element of samples vector 1 and 2 were not sampled values and were not derived from sampled values. Thus, CSNV[n] is set to zero and CSV[n] is not set.

If $c_1 > 0$ and $c_2 = 0$, then we see that data set 1 has information regarding the nth sample point and data set 2 does not. Thus, we set CSNV[n]=SNV1[n] and CSV[n]=SV1[n]. For $c_1 = 0$ and $c_2 > 0$ we set CSNV[n]=SNV2[n] and CSV[n]=SV2[n].

After the above combining operations, we again use linear interpolation to fill in all the possible sampling points where the combined samples number vector is zero. The resulting data set thus has a combined samples vector and a combined samples number vector.

The reasoning behind this procedure is simple. If both data sets have information regarding the nth sample, then we average the samples in hopes of reducing the affect of the additive zero-mean Gaussian noise. If only one data set has information regarding the nth sample then we use the sample from that data set. If neither data set has information regarding the nth sample then we fill in that sample point using linear interpolation.

3.3. DSCA Method Outline

We provide a pseudocode description of the DSCA method in Table 1. Recall that the operation of each node is divided into a signal sampling phase and a signal processing phase and that the network is synchronized. The DSCA method applies specifically to the signal processing phase since it is basically a method to combine distributed sample sets in a decentralized manner and construct a less noisy and more complete sample set at the base station node. Note that the `collectionSize` parameter is the number of data sets the node receives from neighboring nodes before combining the data sets.

Table 1. The DSCA Method

```
DSCA Method (int collectionSize)

while (in signal processing phase) {
    if (just entered signal processing phase and have local
    signal samples) {
        compress local data set;
        localSignal = compressed local data set;
        send out localSignal;
    }
    else {
        numReceivedSignals = 0;
        while (numReceivedSignals < collectionSize) {
            receive data set from neighboring nodes;
            store received data set;
            numReceivedSignals++;
        }
        combinedSignal = localSignal;
        for each stored received data set {
            signal1 = decompress stored data set;
            signal2 = decompress combinedSignal;
            signal3 = combine signal1 and signal2;
            compress signal3;
            combinedSignal = signal3;
        }
        send out combinedSignal;
    }
}
```

4. SIMULATION AND ANALYSIS RESULTS

4.1. Compression Parameters

We first consider the impact of the M, `percentEnergyCapture`, and `numWaveletSamples` parameters on the performance of the DSCA method.

When we discussed signal compression in Section 3, we mentioned that we zero-padded the length of the signal to a power of 2 so that we would have the freedom to choose a large M value. The reason we do this is that in certain cases doing a higher level wavelet transform will allow us to discard more of the transform values, thus achieving better compression. Consider the two waveforms shown in Fig. 3. Signal A is generated by randomly sampling a noisy version of the waveform

$$S_A(t) = \frac{1}{2}(\sin(2\pi x) + \cos(6\pi x))$$

and Signal B is generated from the waveform

$$S_B(t) = \frac{1}{2}(\sin(2\pi x) + \cos(36\pi x)).$$

Both $S_A(t)$ and $S_B(t)$ were corrupted by zero-mean additive white Gaussian noise with variance 0.01 before being randomly sampled.

In Fig. 4, we do a 4-level and 8-level Daubechies-4 wavelet transform on both Signal A and Signal B. In this figure we can see a noticeable difference in the number of levels done in the transform. For both the transforms of Signal B, we notice a set of significant transform values from sample number 64 to 128. Since these larger transform values suggest that they actually contain information regarding $S_B(t)$ and not just noise information, we should definitely keep these values. This means that we can compress the 1024 samples to about 128 samples. On the other hand, if we look at the transforms of Signal A, we see that in the 8-level transform no real significant transform values appear until we get less than sample number 50. Thus, if we only transformed Signal A to level 4, we would have been forced to retain at least 64 samples while transforming it to level 8 would allow us to retain less than 50 samples.

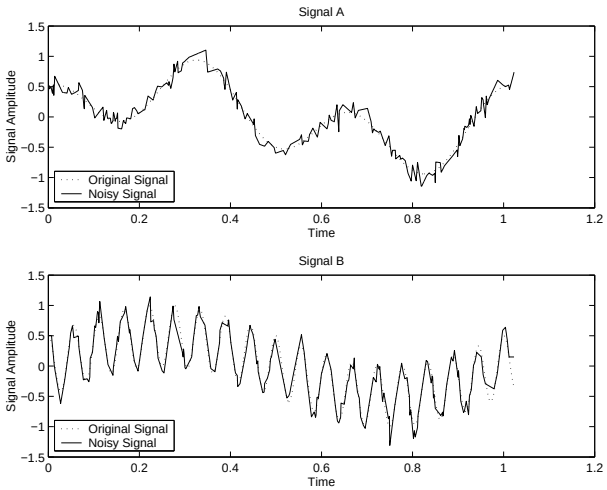

Fig. 3. Signal A (top) and Signal B (bottom) are generated by randomly sampling a noisy version of $S_A(t)$ and $S_B(t)$, respectively.

Another way to see the importance of the choice of M is in doing the energy capture. In Fig. 5 we capture 98% of the energy of the 4-level and 8-level transform of both Signal A and Signal B. We notice that for Signal B, capturing 98% of the energy leaves us with about 128 samples in both cases so transforming that signal to 8 levels was unnecessary. However, for Signal A, capturing 98% of the energy in the 4-level transform leaves us with over 60 samples while in the 8-level transform it leaves us with less than 40. Thus, we see that depending on the information available regarding the type and maximum frequency of the signals that are being observed, the ability to choose the M values may provide for more compression. In general, for a given set of possible sampling points, higher frequency signals will need fewer levels of the wavelet transform. This is because the high frequency nature of the signal will make it show up in the detail signal in fewer levels of the wavelet transform. We show the signals decompressed from the 8-level wavelet transforms with 98% energy in Fig. 6.

The `percentEnergyCapture` parameter is very important in determining how much noise is removed and how much smoothing takes place. In Fig. 7 we show three signals reconstructed from 90%, 98%, and 99.9% of the energy in the signal's 8-level wavelet transform. In the first signal constructed from 90% of the energy, we see that we removed too much information. Not only did we get rid of the noise but we also discarded some information significant to the signal itself. In the last signal constructed from 99.9% of the energy we notice that not enough information was discard and a significant amount of noise is still present. Lastly, the middle signal created from a capture of 98% of the energy gives a decent representation of the original signal and most of the noise has been removed.

The `numWaveletSamples` parameter serves as a way to set the maximum number of samples transmitted by each node. If fewer than `numWaveletSamples` remain after the energy capture then the node can send less than `numWaveletSamples` samples. However, if the number of samples that remain after energy capture is

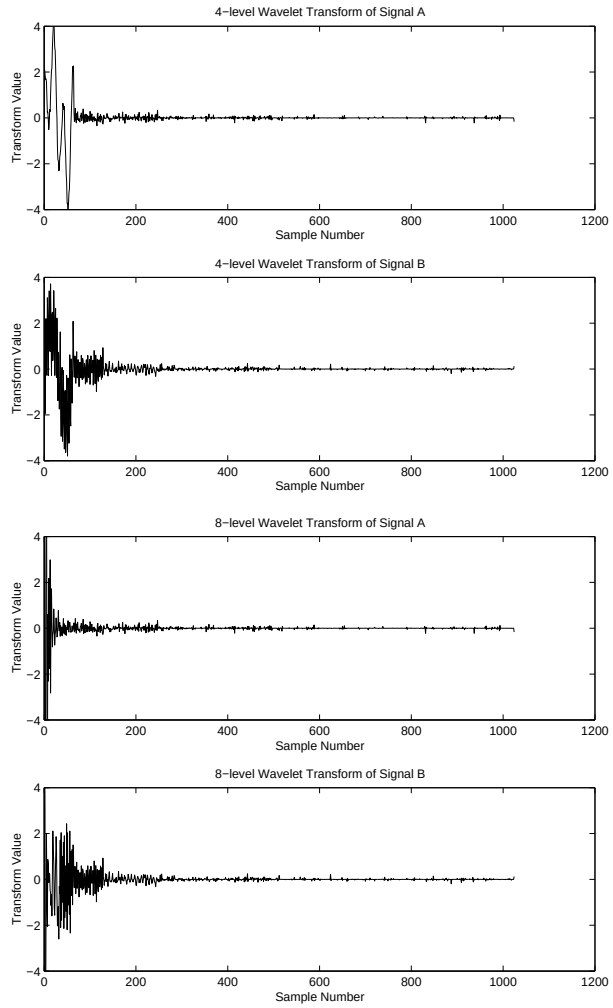

Fig. 4. A 4-level (top two) and 8-level (bottom two) Daubechies-4 wavelet transform of Signals A and B. Notice that for Signal B there is not much more energy compaction to the left as we go from a 4-level transform to an 8-level transform due to the large transform values between samples 64 and 128. However, the energy of Signal A is significantly compacted towards the left as we move to an 8-level transform.

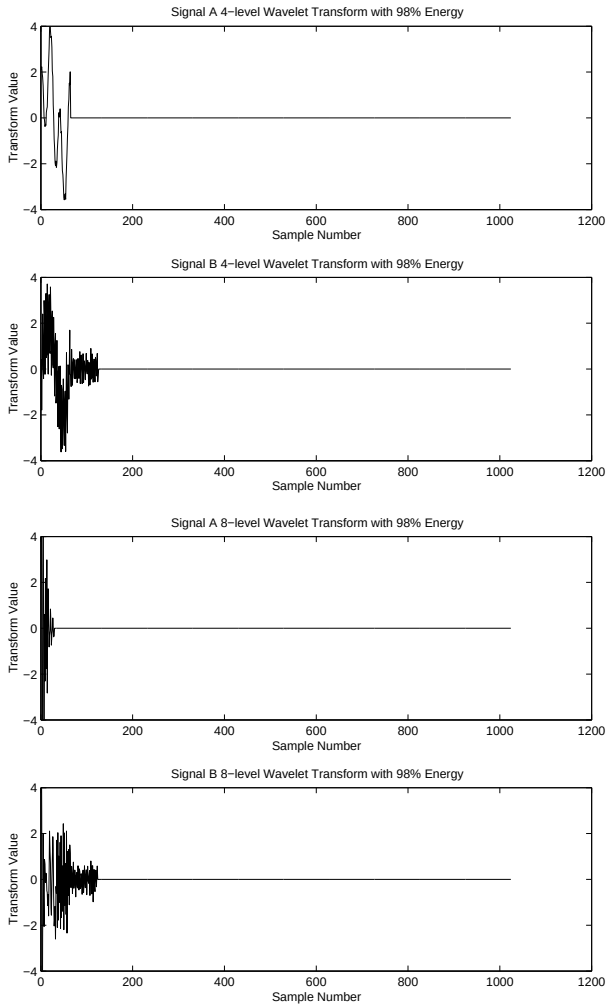

Fig. 5. 98% of the energy is captured in the wavelet transforms of Signals A and B.

greater, then the node will send at most `numWaveletSamples` samples. If such a situation does occur, there may be more distortion in the signal but this parameter does allow the system to control the amount of data that has to be sent.

4.2. MATLAB Simulation Results

We simulate the DSCA method in MATLAB to determine its performance in combining data sets from different nodes. Each data set that we include in this simulation is a completely new data set and thus the combined signal is being given more information regarding the original signal. We would expect the quality of the combined signal to improve as we increase the number of new data sets.

We consider an example signal from [8]

$$g(t) = 20t^2(1-t)^4\cos(12\pi t)$$

shown in Fig. 8. We simulate the performance of the DSCA method using the following parameters:

$$\begin{aligned} M &= 8 \\ \texttt{percentEnergyCapture} &= 99.5 \\ \texttt{numWaveletSamples} &= 75 \end{aligned}$$

and the number of possible sampling points is 1024. We corrupt $g(t)$ with zero-mean additive Gaussian noise with variance 0.01 and consider varying values of p. We average each data point over 25 runs. Note that each data set we incorporate is generated by sampling a noisy version of $g(t)$.

We begin the simulation by taking the first data set and compressing it using an 8-level Daubechies-4 wavelet transform before capturing 99.5% of the energy. We then set to zero all but the first `numWaveletSamples` values in the transform. We next decompress this signal and aggregate it with the next new data set. The combined signal is then compressed in the same way as the first signal. The next new data set is then aggregated with the decompressed version of compressed combined signal. The process continues in this way to include a new data set at each iteration. This simulates the distributed aggregation and compression of a series of independently sampled data sets. The results of this simulation are shown in Fig. 9.

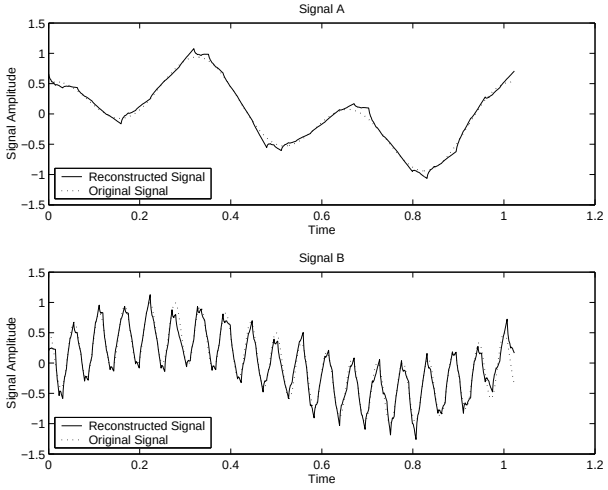

Fig. 6. Signals A and B are decompressed from 98% of the energy of the 8-level wavelet transforms.

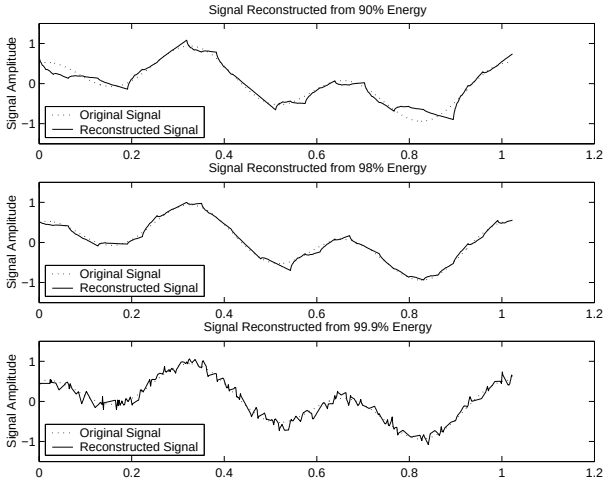

Fig. 7. The signal is decompressed from its wavelet transform with 90%, 98%, and 99.9% of its energy captured. With 90%, too much information is lost and the signal is distorted (top). A good amount of information is removed with 98% energy (middle). Too little data is removed with 99.9% energy and the signal is still quite noisy (bottom).

The figure of merit that we use is the root mean square (RMS) error measure between two signals. This is defined for two signals $\{a_k\}$ and $\{b_k\}$, $k = 1, \ldots, L$ as

$$\text{RMS Error} = \sqrt{\frac{\sum_{k=1}^{L}(a_k - b_k)^2}{L}}$$

From Fig. 9 we clearly see that initially as we combine an increasing number of independently sampled data sets the average RMS error decreases. This is expected since with each data set, we are increasing the amount of information the combined signal has about the original signal. However, the increase in RMS error starting around 25 data sets comes from the fact that the aggregate signal at that point is already a very good representation of the true signal. The data provided by the additional data sets do not offset the loss of information that occurs from the lossy compression stage. As a result, at each step more and more signal information is being taken away. Lastly, we notice that with increasing p, the average RMS error also decreases. A larger p means that each data set contains more samples of the noisy version of $g(t)$ and thus we would expect the RMS error to decrease.

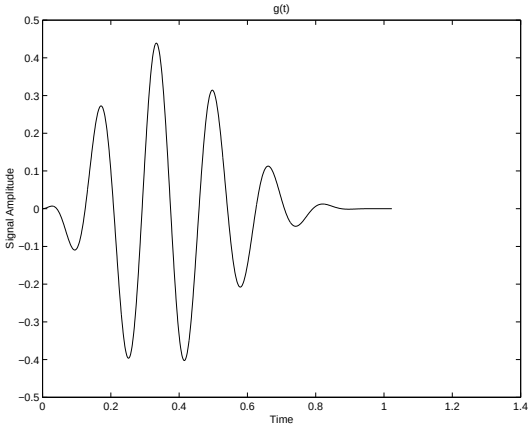

Fig. 8. The source signal $g(t)$ used in the MATLAB simulations.

4.3. Ptolemy Simulation Results

The simulation in Fig. 9 reveals that the DSCA method improves the signal quality as we combine more data sets. However, the simulation may be slightly artificial in that we always incorporate an independently sampled data set at each iteration. Such a setup may not always be the case. In sensor networks it may be the case that a node simply broadcasts its signal data and aggregates whatever signal data it receives from neighboring nodes. As a result, a node may repeatedly incorporate the same data and not always be adding new information to the combined signal.

To understand how the DSCA method behaves in such a situation, we implement the method in a sensor network modelled with the VisualSense configuration of Ptolemy II [9]. A screen shot of the simulation environment is presented in Fig. 10. We see 16 sensor nodes arranged in a grid formation with a target signal source in the upper left hand corner of the network. The circle surrounding each sensor node represent the transmission range of the node and the large circle centered at the signal source is the transmission range of the source. Each sensor node within the transmission range of the source is able to sample the signal. The goal is to communicate the signal observations to one of the Tier 2 nodes at the lower right hand corner of the network. In Fig. 11 we show the noisy signal observed by a node in the source signal transmission range and the signal communicated to a Tier 2 node.

The communication between nodes is modelled as follows. At the beginning of the signal processing phase, each node that has a sampled data set of the signal will compress the data set and send the data to its communication module where it is queued. A Poisson clock controls the queue and at each clock tick an element of the queue is broadcast to the node's surrounding neighbors. The Poisson clock ticks at times determined by a Poisson processes with mean `meanTxTime`. Each node in the network will also be collecting data sets received from surrounding nodes. A node will collect at least `collectionSize` data sets before running the DSCA method for aggregation and compression. It will combine the received data sets along with its own data set. After the DSCA method, the node will send the combined data set to the communication mod-

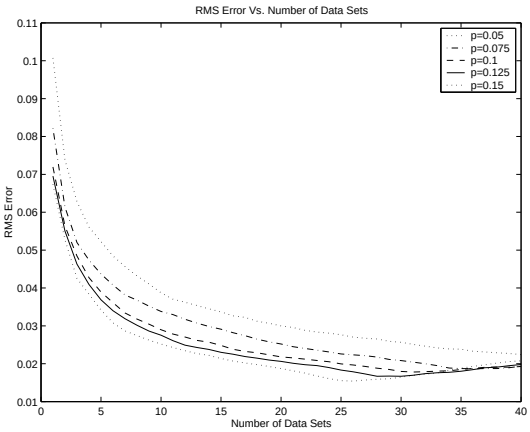

Fig. 9. MATLAB simulation results evaluating the performance of the DSCA method. We clearly notice that initially as more data sets are included, the RMS error decreases. The error eventually starts increasing as any new information provided by the additional data sets does not offset the signal information lost during the compression stage.

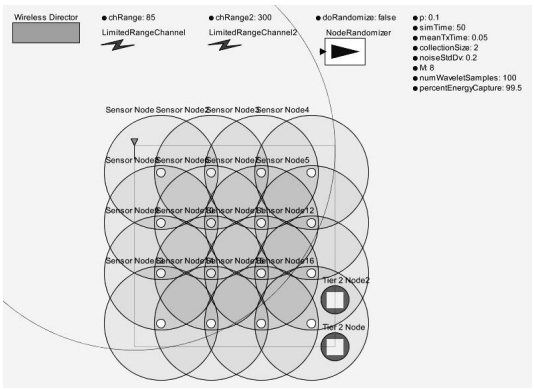

Fig. 10. A gray scale screen shot of the simulation environment in Ptolemy II.

ule. The transmission queue is cleared when the network enters the signal sampling phase. Notice that the nodes do not maintain any information regarding the origins of the data sets they receive. Each node simply receives a collection of data sets and aggregates the information.

In the following Ptolemy II simulations, we keep the following parameters constant.

$$\begin{aligned} M &= 8 \\ \texttt{numWaveletSamples} &= 75 \\ \texttt{percentEnergyCapture} &= 99.5 \\ p &= 0.1 \\ \texttt{collectionSize} &= 2 \end{aligned}$$

We also keep the node communication range 85 distance units to give each node the ability to communicate with its four nearest neighbors. The simulation data is gathered by a Tier 2 node that is placed in the transmission range of a node that is in the broadcast domain of a node hearing the signal transmissions. That is, if node A is in the range of the signal source and node B is in the transmission range of A but not within the signal source range, then the Tier 2

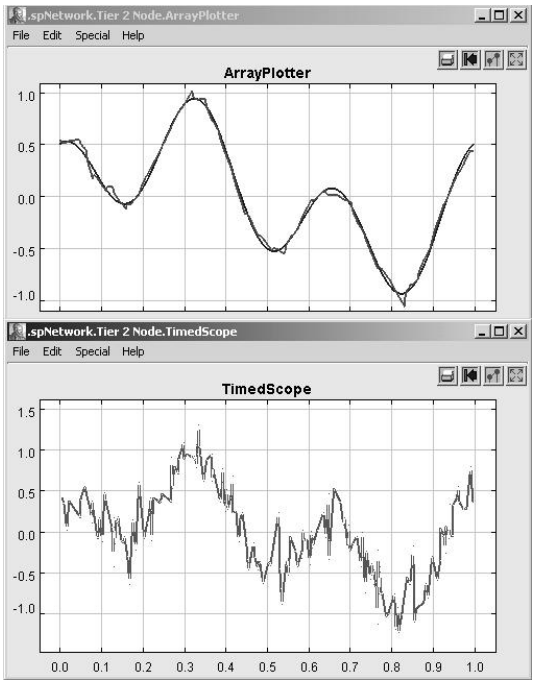

Fig. 11. A gray scale screen shot of two display windows from the the Ptolemy II simulation. The top window shows the original signal waveform overlayed with the signal waveform available at the Tier 2 node after being processed by the DSCA method. The bottom figure is the noisy waveform observed by a node within the transmission range of the signal source.

node would be placed in the transmission range of B but not in the broadcast domains of A or the source signal. Furthermore, this Tier 2 node is placed as close as possible to the diagonal line connecting the signal source and the and lower right hand corner node. This node placement is used since for the chosen node to node communication model, signal data diffuses slowly through the network. Thus, by placing the Tier 2 node close to the boundary of the signal source transmission range guarantees that the Tier 2 node will receive signal data.

In Fig. 12, we consider the RMS error of the signal received at the Tier 2 node versus the transmission range of the source signal. We set $\texttt{meanTxTime}= 0.2$. The trend seen when varying the transmission range from 150 units to 250 units is expected since we are increasing the number of nodes within the range of the signal source from 3 nodes to 8 nodes (at a range of 200 there are 6 nodes in the transmission range). As we increase the number of nodes in the signal range, the network has more independently sampled data sets and thus it is reasonable as seen in Section 4.2 that the Tier 2 node can get an increasingly better view of the source signal. However, at source signal ranges 300 units and 350 units with 13 nodes and 15 nodes, respectively, we witness an increase in RMS error. The reason for this increase is because as we increase the number of nodes in the network we also increase the amount of traffic in the network since more nodes are sending in their own signal observations. With increased traffic we begin to see the problem of over processing the signal. Consider, for example, a signal that has been processed by the DSCA method and is the combination of all independently sampled data sets in the network. If we send this signal back into the network and continue processing it, then each time we do lossy compression on the signal a little bit of the signal is being stripped

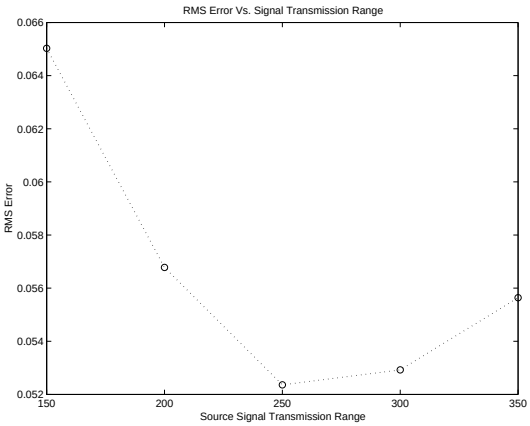

Fig. 12. A plot of RMS error versus the transmission range of the source signal.

away. Without any new signal information being added, the original underlying signal will start becoming distorted. In the type of node to node communication setup that we use for the simulations, it is very possible for the signal to be over processed since the nodes do not consider the origins of the received data sets.

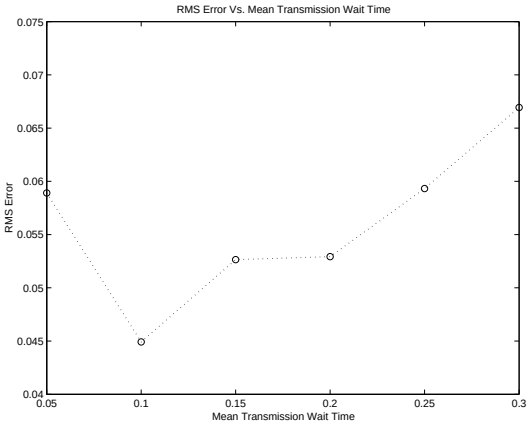

Fig. 13. A plot of RMS error versus the average transmission wait time `meanTxTime`.

We consider varying the `meanTxTime` parameter to get another view of this issue. By increasing `meanTxTime`, we decrease the amount of traffic in the network since it takes longer for a node to send its data. We set the source signal transmission range to 300 distance units. In Fig. 13 we see a plot of RMS error versus `meanTxTime`. The expected upward trend is seen as we vary the mean transmission wait time from 0.1 to 0.3. With an increase in `meanTxTime` we get a decrease in network traffic and we effectively decrease the number of independently sampled data sets that can be combined before reaching the Tier 2 node. However, from a mean wait time of 0.05 to 0.1 we see a decrease in the error. This also reveals that if there is too much network traffic, we get an over processing of the signal and can actually decrease the quality of the signal seen at the Tier 2 node. Thus, the key insight from the simulations done in Ptolemy II is that if the network communication methods do not guarantee the constant infusion of new data sets, then the system designer must be conscious of the problem of possibly over processing the signals.

5. CONCLUSION

In this work we have presented the DSCA method for the distributed aggregation and compression of decentralized low resolution samples of a signal source. Simulations have shown that the DSCA method significantly reduces the RMS error between the reconstructed signal and the original signal as we combined an increasing number of independently sampled data sets. The method was also implemented in the Ptolemy II network simulation environment and is shown to perform under more realistic network communication protocols. This technique can be likened to taking many low quality snap-shots of the signal and combining them in a distributed manner to form a high quality picture of the image.

For future work, we would like to consider the effects of different types of wavelet transforms on the performance of the DSCA method. As well, another important area is the effects of phase shifted signals. For situations where the signal source produces a slower propagating signal, such as seismic vibrations or acoustic signals, we have the problem of each node seeing the same signal with different phase shifts. Some preliminary results (not presented here) suggest that if each node's low quality signal sample has enough information for the data sets to be aligned, then the DSCA method will still perform well. However, more study is needed to understand how well the DSCA method will carry over to phase shifted signals.

6. REFERENCES

[1] J. Elson, L. Girod, and D. Estrin. Fine-Grained Network Time Synchronization using Reference Broadcasts. In *Proc. Fith Symposium on Operating Systems Design and Implementation*, Boston, MA, 2002.

[2] H. G.Feichtinger, C. Cenker, and H. Steier. Fast Iterative and Non-Iterative Reconstruction Methods in Irregular Sampling. In *Conf. ICASSP*, Toronto, Canada, 1991.

[3] H. Wang, D. Estrin and L. Girod. Preprocessing in a Tiered Sensor Network for Habitat Monitoring. *EURASIP Journal on Applied Signal Processing*, 4:392-401, 2003.

[4] X. Wang and H. Qi. Acoustic Target Classification Using Distributed Sensor Arrays. In *Proc. International Conference on Acoustic, Speech, and Signal Processing (ICASSP)*, Orlando, FL, 2002.

[5] Y. Tian and H. Qi. Target Detection and Classification Using Seismic Signal Processing in Unattended Ground Sensor Systems. In *Proc. International Conference on Acoustic, Speech, and Signal Processing (ICASSP)*, Orlando, FL, 2002.

[6] M. Moran, S. Ketcham, and T. Anderson. Virtual Proving Ground for Networks of Seismic Unattended Ground Sensors. *24th Army Science Conference (ASC)*, Orlando, Fl, 2004.

[7] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton and J. Zhao. Habitat Monitoring: Application Driver for Wireless Communications Technology. In *Proc. 1st ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, San Jose, Costa Rica, 2001.

[8] J. S. Walker. A Primer on Wavelets and their Scientific Applications. Chapman & Hall/CRC, Boca Raton, FL, 1999.

[9] P. Baldwin, S. Kohli, E. A. Lee, X. Liu, and Y. Zhao. Modeling of Sensor Nets in Ptolemy II. In *Proc. Information Processing in Sensor Networks (IPSN)*, Berkeley, CA, 2004. (see also http://ptolemy.eecs.berkeley.edu/ptolemyII/).