Toward Energy Efficient Multistream Collaborative Compression in Wireless Sensor Networks

Tommy Szalapski and Sanjay Madria
Department of Computer Science, Missouri S&T, Rolla, MO 65401, USA
T.M.Szalapski@mst.edu and madrias@mst.edu

***Abstract* -Wireless sensor networks possess significant limitations in storage, bandwidth, and power. This has led to the development of several compression algorithms designed for sensor networks. Many of these methods exploit the correlation often present between the data on different sensor nodes in the network; however, correlation can also exist between different sensing modules on the same sensor node. Exploiting this correlation can improve compression ratios and reduce energy consumption without the cost of increased traffic in the network. We investigate and analyze approaches for compression utilizing collaboration between separate sensing modules on the same sensor node. The compression can be lossless or lossy with a parameter for maximum tolerable error. Performance evaluations over real world sensor data show increased energy efficiency and bandwidth utilization with a decrease in latency compared to some recent approaches for both lossless and loss tolerant compression.**

Keywords - wireless sensor network; real-time; collaborative; compression;

I. INTRODUCTION

Wireless sensors are used to collect and transmit data in a wide variety of applications. Many such applications utilize sensor nodes that collect several different streams of data on different sensing modules on the same sensor node. For example, sensor nodes in the Great Duck Island project [1] and an Intel Berkeley Labs experiment [2] were used to collect temperature, humidity, light intensity, and more. Even applications that primary just sense one thing often send multiple streams of data from the same sensor. For example, ZebraNet [3] tracked locations of zebras sending two streams of data for the GPS readings (easting and northing) and some metadata such as voltage and count of satellites in range of the GPS sensor.

It is well known that wireless sensor networks possess significant limitations in processing, storage, bandwidth, and power. This has, naturally, led to the development of many compression algorithms specific to sensor networks. Many of these algorithms rely on the data readings from a single sensor being correlated to previous readings on that same sensor (temporal locality) [4][5][6]. Others rely on correlations between similar data streams on other sensor nodes (spatial locality) [7][8][9][10]. Correlation can also exist between different streams of data collected on the same sensor node; however, very little work has yet been done which exploits this correlation.

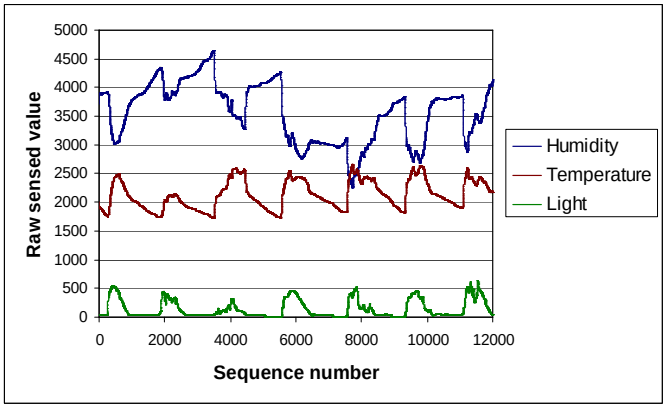

Figure 1 Multistream sensor readings

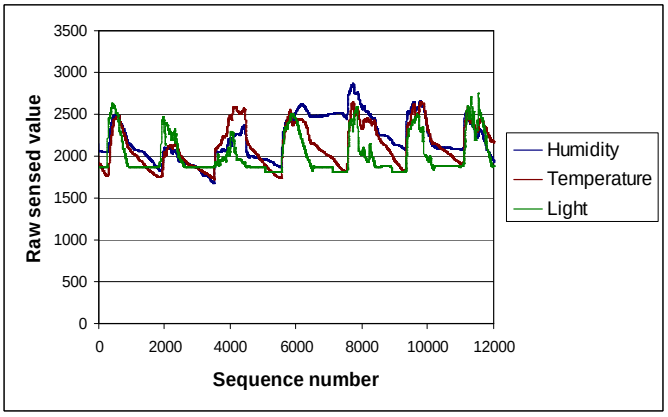

Figure 2 Scaled multistream sensor readings

To illustrate this correlation, Figure 1 shows values from 12,000 readings of temperature, humidity, and light intensity sensors on a single sensing node taken from the Intel Lab dataset. Figure 2 shows those same values scaled with the simple linear transformations shown in Equation 1 where h_n is the nth humidity reading and h_n' is the scaled value. Similarly, t_n and l_n are for the temperature and light intensity, respectively along with their scaled notation. Clearly some benefits could be gained by leveraging the correlation between the different data streams.

$$\begin{aligned} h_n' &= 4000 - 0.5h_n \\ t_n' &= t_n \\ l_n' &= 1800 + 1.5l_n \end{aligned} \quad (1)$$

COLLABORATECOM 2014, October 22-25, Miami, United States

DOI 10.4108/icst.collaboratecom.2014.257289

In this paper, we present TinyPack-Collaborative (TinyPack-C), a lightweight compression algorithm leveraging the temporal correlation within each stream and the correlation between multiple streams of data on an individual sensing node. TinyPack-C is based on the initial code set presented in [6] and extended to include collaboration between the multiple streams from the various sensors on the same sensing node. Collaboration is computed based on a rolling linear regression scheme requiring constant time memory use and processing for each correlated pair of sensed values.

If some loss is tolerable in the data, compression is enhanced by first performing a modified version of the jumping baseline transformation introduced in [11] which converts the stream into a step function. The rolling linear regression is then applied to the flattened streams. The maximum tolerable error can be configured low for simply removing noise from the data or high if the application is not concerned with low variation in the data.

We present and analyze compression schemes for both lossless compression and loss tolerant compression with a configurable maximum error. We compare both varieties against state of the art compression methods. For the lossless case, we compare against the original TinyPack algorithm, LEC [5] and S-LEC[12]. We compare our lossy compressor with LTC [13] and the single sensor jumping baseline approach [11]. Simulations using TOSSIM [18] were done over several real life datasets covering a wide variety of sensor applications.

In summary, this paper makes the following contributions:

- Novel algorithms for lossless compression leveraging collaboration across multiple streams on a single sensor node
- Additional algorithms for lossy compression with a configurable upper bound for error
- Lightweight mechanisms for computing correlation between signals
- Detailed analysis over several real world datasets
- Methods for performing mathematical operations and aggregation on the compressed data without first decompressing the data
- Analysis of effects of a simple signal reconstruction method on measured error

II. Related Work

A. S-LEC

S-LEC, a lossless data compression scheme, is proposed in [12]. S-LEC begins with the static set of codes used in LEC [5] to represent delta values in a data stream. In LEC, each reading, the previous value is subtracted from the current value and the resulting delta value is coded based on a static table of codes derived from those used in JPEG compression. Smaller delta values have shorter codes. For S-LEC, codes that are the same length are said to be in the same group and two bits are prepended to each value noting whether the current delta value is in the same, one higher, one lower, or any other group as the previous delta value. This enables reducing the size of the prefix come and improves the compression ratio when data is changing in a consistent fashion.

B. TinyPack

Another lossless method is presented in [6], TinyPack initially uses a similar set of static codes for its compression, but the codes were optimized for wireless sensor data instead of JPEGs. Those codes are then dynamically modified either by counting the frequency of each value or by approximating those frequencies using a rolling average and standard deviation. The initial set of codes used in TinyPack-Init is shown in Table I and forms the basis on which the compression in this work is built.

Table I Static Codes

prefix	suffix range	values
1	n/a	0
01	0...1	-1.1
001	00...11	-3,-2,2,3
0001	000...111	-7,...,-4,4,...,7
00001	0000...1111	-15,...,-8,8,...,15
000001	00000...11111	-31,...,-16,16,...,31
0000001	000000...111111	-62,...,-32,32,...,63
00000001	0000000...1111111	-127,...,-64,64,...,127

Except in the case of 0, the last bit of the suffix is the sign bit. For example, if the current reading was 3 higher than the previous reading, a delta value of +3 would be transmitted as 00110. A delta value of -4 would be encoded as 0001001. Note that in [6] the sign bit was at the beginning of the prefix, but computing mathematical operations on the compressed data is easier if the sign bit is moved to the end.

C. LTC

In [13] a lossy compression scheme is introduced that approximates the data stream by a sequence of linear segments. As the data is collected by the sensor, the algorithm fits a line to the data as long as the line can be defined such that no point in the transformed data exceeds a maximum error bound. When a data point is sensed that cannot be fit to the line without exceeding the allowed error, that line is transmitted and a new line starts. The algorithm is effective but does introduce additional latency since the data is not transmitted until the sensed reading that necessitates a new line.

D. Jumping Baselines

The jumping baseline approach in [11] approximates the data stream as a discrete step function which can be reconstructed to a linear function similar to the one generated by LTC at the sink. Any time a sensed value is outside the maximum tolerable error away from the current baseline, a new baseline is selected. The possible candidate baselines are selected from multiples of the maximum error such that the new value can be expressed as the number of baseline jumps above or below the previous baseline. The new baseline is also selected as far in the direction the data has been trending as possible without violating the maximum tolerable error. This process is described in more detail in section 0 and forms the basis on which our lossy compression is built.

III. BACKGROUND

A. Temporal locality

Data from wireless sensor networks generally exhibits temporal locality (data values from the same stream are correlated to values that are close together in time). Any type of data stream which changes in a continuous fashion will be temporally located such as humidity, position, light intensity, water level, etc. In fact, it can be demonstrated that any sensor stream sampled at non-random intervals will either generate temporally located data or random noise.

Consider an arbitrary sensor sensing a stream of values $\{v_1, v_2, \ldots, v_{2N}\}$ sensed at times $\{t_1, t_2, \ldots, t_{2N}\}$ where N is an integer. Assume that the values are not correlated. Then sampling at $\{t_1, t_3, \ldots, t_{2N-1}\}$ and $\{t_2, t_4, \ldots, t_{2N}\}$ would yield completely different values. Thus, offsetting the sample period would generate entirely different data. Therefore, application with time-based sampling which did not exhibit temporal locality must be sampling random noise. Excluding such applications we can assume that successive readings at each sensor will be correlated. Delta compression (storing the data as the change in value from the previous reading) would then increase the frequency of certain values thus increasing the compressibility of the data.

Naturally this does not apply to event driven sampling (where time between samples is random) such as a sensor that measures the speed once for each passing automobile. These applications do not necessarily exhibit temporal locality and were not included in this study.

The previously sensed value in each sensed stream can then be used as a baseline for compressing the value of the next sample in the stream. For lossless compression, the value can be transmitted as the difference between the current sensed value and the previous value (the *baseline value*). For lossy compression, the data can be approximated using the baseline value until the current value differs from the baseline value by more than the upper limit for tolerated error.

B. Collaborative compression

In the case of collaborative compression, one sensed stream serves as the baseline for one or more of the other sensed streams on the same sensor. The data from this *baseline stream* is compressed leveraging temporal locality as discussed in the previous section and the data from the correlated streams are encoded based on the difference from some linear function of the baseline stream referred to as the *baseline function*. As with the single stream compression of the baseline stream, the lossless case would require that a delta value be sent every time the sensor samples data while the lossy case can use the baseline function as the approximated values for the compressed stream until the value is above or below the baseline function by more than the maximum tolerable error. The algorithm is shown in more detail section 0.

C. Measuring error

For the lossy compression, we consider a parameterized maximum tolerable error percentage E_{max}. Instead of reporting every value exactly as sensed, if a value deviates from its baseline less than E_{max}, the baseline value can be used instead. This allows for much greater compression while keeping the error bound by the tunable maximum. This parameter can be adjusted based on the application need, i.e., in real-time, but can tolerate some error (lossy), or non-lossy, but can tolerate some latency.

A common method of measuring error, E, between a reported value, V_R, and the actual value V_A, is shown in Equation 2.

$$E = \frac{|V_A - V_R|}{|V_A|} \tag{2}$$

Unfortunately, that measure does not work well for many kinds of sensor data when introducing error because the error varies wildly when working with values near zero.

Consider a sensor which reported relative humidity readings with a maximum error of +/- 1. Table II shows several possible actual readings and their approximated values within 1 of the actual value. Also shown is the calculated error using the formula shown in Equation 2.

Table II INCONSISTENT ERROR MEASURE

actual value	*approximated value*	*calculated error*
48	49	2.08%
14	15	7.14%
2	3	50%
0	1	undefined

In practice, the best way to set an upper bound for error would be to explicitly set the bounds in terms of the scale. For example, when set by the end user, the tolerable error for a temperature reading could be +/- 1°C. For analysis, however, it is useful to have a method of normalizing the error to a percentage. Another common method of measuring error is to divide the difference by the maximum range. The formula could use the maximum range of the sensor; however, since this range can be very large compared to the actual sensed range, the error percentages would be artificially low. For our analysis we use the maximum range of actual sensed values as the denominator for the error normalization (see Equation 3).

$$E = \frac{|V_A - V_R|}{V_{MAX} - V_{MIN}} \tag{3}$$

Table III shows the calculated error for the same data assuming the humidity measurements ranged from 0 to 49. This is a much better error measure for the work presented in this paper.

Table III CONSISTENT ERROR MEASURE

actual value	*approximated value*	*calculated error*
48	49	2.00%
14	15	2.00%
2	3	2.00%
0	1	2.00%

D. Jumping baseline compression

For our lossy compression algorithm, we begin with the jumping baseline compression introduced in [11]. The values in the stream are compressed to a step function by choosing a baseline value for a sensed value and only changing the baseline when the current sensed value differs from the baseline by more than the maximum tolerable error. The values selected as baselines are in the form kE where k is any integer and E is the maximum integer error that can be tolerated in a stream while remaining within the maximum error percentage E_{max}.

The initial baseline is selected by choosing the candidate baseline closest to the first value sensed in a stream. So for a sensed value v the baseline B would be selected as shown in Equation 4. Adding 0.5 and truncating with the floor function is done as an efficient method of rounding.

$$k = \left\lfloor \frac{v}{E} + 0.5 \right\rfloor \qquad b = kE \tag{4}$$

When a sensed value differs from the current baseline by more than E, a new baseline must be selected. Note that there will be two candidate baselines that would be within E of the new value. The algorithm chooses the baseline based on which direction the data is trending. A data stream can be in one of three states: trending up, trending down, or staying somewhat constant. If data is trending either up or down, then the next baseline should be selected as far in the direction the data is trending as it can be within the error bounds. If the data is remaining relatively constant, then the next baseline should be selected as close to the current value as possible. The state is determined by tracking whether the new baseline is above or below the previous baseline for two jumps. If both jumps were in the same direction, the data is trending either up or down depending on the direction of the jumps. All that needs to be cached is the previous value and the previous jump direction. The additional computation is also trivial. For example, Table IV shows an example of a light sensor with a maximum error set at +/- 10 lux.

Table IV BASELINE COMPRESSION EXAMPLE

Seq no	Sensed value	Last value	Last jump	This jump	Baseline
1	242	--	--	--	240
2	253	242	--	up	250
3	261	253	up	up	270
4	276	261	up	--	270
5	284	261	up	up	290

Initially, the baseline is selected as close as possible to the actual sensed value. When the upward trend is established at sequence number 3, the baseline is selected as high as possible while remaining within the error tolerance of +/- 10. Then as the data continues to trend upward, the baseline does not require as many jumps while remaining within the maximum tolerable error. This process is shown in detail in Algorithm 1.

Algorithm 1 CheckReading(v, p, S, d)

Objective: Check current reading, select next baseline
Input: Sensed value v, previous baseline B, max difference E, previous jump direction d
Output: New baseline (reported value) B

```
If |p – v| > E
    B := floor(v/E + 0.5)
    If v > B And d == UP
        B := B + E
    Else if v < r And d == DOWN
        B := B – E
    End If
    If v > p
        d := UP
    Else
        d := DOWN
    End If
    p := B
Else
    B := p
End If
```

IV. OUR MULTISTREAM COMPRESSION APPROACH

A. Rolling correlation

A common simple method of approximating one data stream with another is to use a linear least squares approximation. The first stream is translated using a linear function in the form $Y = aX + b$ into an approximation of the second stream in such a way as to minimize the amount of error between the approximated stream and the actual stream. Computing full least squares regression is far too computationally complex to run on a sensor every time a new value is sensed; however, the correlation can be computed incrementally such that only a few calculations need to be made after each sample while still maintaining accurate correlation values.

Also, the correlation is not necessarily the same for the entire run of the sensor network so some decay should be introduced in the correlation equation such that the most recent data contributes a higher weight to the correlation and older data contributes less. Such decaying rolling statistics have been used many times for other applications [6][14][15]. Here we refine the rolling least squares to optimize for simplicity of calculation for the sensor networks.

A common method for calculating the slope and intercept of the regression line (correlation function) $Y = aX+b$ is shown in Equation 5 where σ_X is the standard deviation of X, $E(X)$ is the expected value (mean) of X, and r is the Pearson Correlation of X and Y.

$$b = r\frac{\sigma_Y}{\sigma_X} \qquad a = E(Y) - bE(X) \tag{5}$$

The standard deviation of a variable can be expressed in terms of the expected values of the variable and the square of the variable as shown in Equation 6.

$$\sigma_X = \sqrt{E(X^2) - (E(X)^2)} \tag{6}$$

The Pearson Correlation coefficient is also commonly expressed in those terms as shown in Equation 7.

$$r = \frac{E(XY) - E(X)E(Y)}{\sigma_X \sigma_Y} \tag{7}$$

Combining equations 5, 6, and 7 we can derive Equation 8.

$$\begin{aligned} b &= \frac{E(XY) - E(X)E(Y)}{\sigma_X \sigma_Y} \frac{\sigma_Y}{\sigma_X} \\ &= \frac{E(XY) - E(X)E(Y)}{(\sigma_X)^2} \\ &= \frac{E(XY) - E(X)E(Y)}{E(X^2) - (E(X)^2)} \end{aligned} \tag{8}$$

Since *E(X)* is simply the sum of *X* divided by the count of samples, if a running total is kept for *X*, *Y*, *XY*, and X^2, then the correlation function can be updated incrementally at each sensed value with a computational complexity of *O*(1).

To allow more recent samples to have a greater impact on the correlation function we introduce a window size *W* over which to compute the statistics. We use the notation X_W to indicate the average of *X* over the window *W*. At each sensed value of X_i, X_{Wi} is recomputed using Equation 9 so that the effect of older samples on the value of X_W slowly decays toward zero. We use $[XY]_W$ and $[X^2]_W$ for the averages of *XY* and X^2 respectively.

$$X_{W_i} = \frac{W-1}{W} X_{W_{i-1}} + \frac{1}{W} X_i \tag{9}$$

In practice, if the current number of samples *N* was less than *W*, then *N* was substituted for *W* in the equations. In that case X_W is the actual mean of the current samples of X_1 through X_N.

This leads us to the final equations for rolling least squares calculations for the correlation function used in this work shown in Equation 10.

$$\begin{aligned} b &= \frac{[XY]_W - X_W Y_W}{[X^2]_W - (X_W)^2} \\ a &= Y_W - bX_W \end{aligned} \tag{10}$$

The mean square error (*MSE*), a measure of the average deviation from the correlation function, can also be computed on the fly in a similar fashion. The general equation for calculating mean square error over variables *X* and *Y* given the correlation function defined by some *a* and *b* is shown in Equation 11.

$$MSE = \frac{\sum_i^N \left((Y_i - (aX_i + b))^2 \right)}{N} \tag{11}$$

This can be expanded and shown in the same form as the other equations used here as shown in Equation 12.

$$\begin{aligned} MSE &= \frac{1}{N} \sum_i^N \left((Y_i - aX_i - b)^2 \right) \\ &= \frac{1}{N} \sum_i^N \left(Y_i^2 - aY_i X_i - bY_i - a^2 X_i^2 + abX_i + b^2 \right) \\ &= [Y^2]_W - a[XY]_W - bY_W - a^2 [X^2]_W + abX_W + b^2 \end{aligned} \tag{12}$$

The coefficient of determination, usually written as R^2 and used to measure the strength of the correlation, can also be computed incrementally. R^2 is simply the square of the *r* value from Equation 7 and is shown in Equation 13.

$$R^2 = \frac{([XY]_W - X_W Y_W)^2}{([X^2]_W - X_W^2)([Y^2]_W - Y_W^2)} \tag{13}$$

B. Collaborative correlation

The above formulas can be used to dynamically track the correlation function between two streams as well as to periodically reevaluate which streams are correlated with which other streams.

Since the correlation function is computed in real time as the data stream is sensed, the correlation is built on the previous values and is not affected by the current sensed value until that value has been transmitted. This enables the calculations to be done on the sink side as well the data is being decoded so that the correlation function is known without the need to transmit the correlation function across the sensor nodes wireless channel. This helps to reduce the total amount of bandwidth required by the application.

For the lossy case, the correlations must be computed after the values have been truncated to the baselines otherwise the sink side would not have the same data on which the correlations were built and would thus be unable to decode the stream unless the correlation functions were transmitted periodically along with the data.

A correlated stream can then encode its values as offsets from its correlation function of its baseline stream. A higher R^2 value indicates a higher correlation and therefore serves as a good metric for which stream to choose as a base for which other streams.

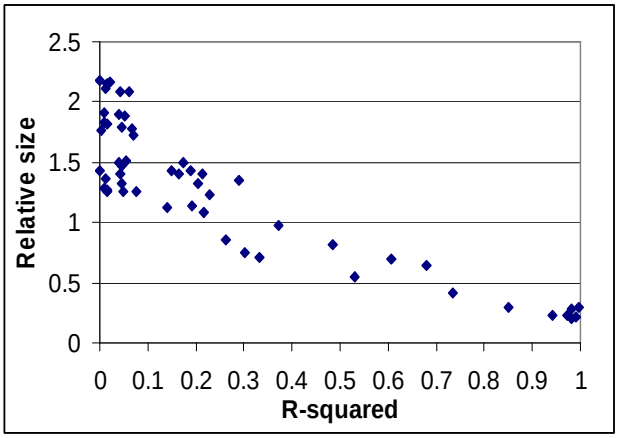

Figure 3 Compressed size for correlated pairs by R^2 value

The computational complexity for computing the correlation for every pair of streams is on the order of $O(S)$ where S is the number of streams. The number of streams on a single sensor node tends to be relatively low (the Great Duck Island weather dataset [1] had 12 which is the highest count of any of the datasets studied here). Even though the number of streams is low, the computation is still too heavy to be ideal. However, while the correlation function can be very dynamic, the sets of correlated streams tend to be rather static, i.e., if some set of streams is found to be correlated, they are typically correlated for the entire run of the dataset. The R^2 values then need not be recomputed every time but only on occasion. Also in many applications, the computations can be done on the sink (which typically has much more processing power) and the correlated sets communicated back through the network. In our experiments, we recomputed the correlation sets every $10W$ samples (where W is the window size of the correlation functions).

To determine when to apply a correlation function, we analyzed each pair of streams on the sensor nodes from the Great Duck Island weather dataset. Figure 3 shows the R^2 value of each pair along with the compressed size using the correlation function divided by the compressed size using just the TinyPack-Init codes. If two streams were not correlated, then adding the correlation function as the baseline for a stream naturally required more bits to transmit the data. Most of the pairs of streams with an R^2 value greater than 0.25 had compression gains when using the correlation function. In our algorithm, any pair of streams with a measured R^2 value greater than 0.25 is defined as a correlated set.

If two streams are correlated to only each other, the one with the lower index is chosen as the baseline stream. If three or more are correlated to each other, then the R^2 values are summed for each pair a stream is in and the stream with the highest R^2 sum is selected as the baseline stream. For example, consider a sensor node sensing temperature (T), humidity (H), and light intensity (L) with the R^2 values for the stream pairs measured as shown in Equation 14. The humidity stream would be selected as the base stream since it has the highest sum of R^2 values as shown in Equation 15.

$$R^2_{T,H} = 0.68 \quad R^2_{H,L} = 0.62 \quad R^2_{T,L} = 0.53 \qquad (14)$$

$$\begin{aligned} sum_T &= R^2_{T,H} + R^2_{T,L} = 1.21 \\ sum_H &= R^2_{T,H} + R^2_{H,L} = 1.32 \\ sum_L &= R^2_{T,L} + R^2_{H,L} = 1.15 \end{aligned} \qquad (15)$$

V. Experimental Set Up

A. Datasets

The datasets used for simulation were pulled from a wide variety of domains, which utilize wireless sensor networks including environment monitoring, animal tracking, vehicle-to-vehicle communication, and smart phone accelerometers. All are from publicly available real deployments of wireless sensor networks.

The Great Duck Island (GDI) [1] experiment deployed sensor nodes in and around the burrows of Leach's Storm Petrels. 32 sensors were deployed monitoring sensor voltage and various types of temperature, humidity, barometric pressure, and solar radiation. Data was analyzed to provide knowledge about the nesting conditions and behaviors of the birds. Strong correlations were observed between temperature, humidity, and solar radiation. Barometric pressure was also somewhat correlated.

For the Intel Berkeley Labs (Lab) [2] deployment, 54 sensor nodes were configured inside a laboratory and used to transmit readings of temperature, humidity, light intensity, and voltage. Temperature, humidity, and light were all correlated, but voltage was not correlated to any other stream.

The ZebraNet project (ZNet) [3] tracked Kenyan zebras generating sensor readings of GPS position and some contextual data about the sensor nodes themselves such as the voltage, count of connected satellites, and horizontal delusion of precision. The sensors were attached to the Zebras and data was used to analyze the social patterns of the animals.

The GATech Vehicular dataset (GATech) [16] was obtained testing a vehicle-to-vehicle network while the vehicles were in motion. Data streams included location, altitude, and speed of the vehicles along with bytes sent and received, signal strength, and noise.

The CenceMe project [17] examined the performance of a system combining off-the-shelf sensor-enabled mobile phones and the automatic sharing and aggregation of the data using social networking applications. Data was gathered by 22 different users and contained readings from the various sensors on the mobile phones including the Bluetooth, GPS, and accelerometer sensors.

B. Implementation

The algorithms were implemented in TOSSIM [18] on simulated MicaZ [19] motes. Experiments were done to show the impact of collaborative compression between the streams on bandwidth usage, energy consumption, and latency. PowerTOSSIM [20] was used to simulate the energy usage for each of the algorithms.

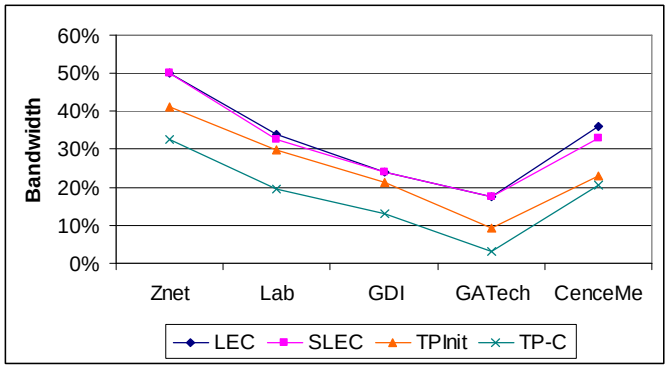

Figure 4 Bandwidth for lossless algorithms

Latency was measured by implementing the algorithms on TelosB motes [21] sending to a base station connected to a notebook computer. The data was stored on the sensor nodes before the experiments and was compressed and transmitted as if the sensors had sensed it. Thus, the time required for actually sensing the data was not included in the experiments; however, since those times are not related to the compression method used, the data would be uninteresting and would approximately be constant for each dataset.

Lossy compression was done four times for each algorithm and dataset. Maximum error was set to 5%, 2%, 1%, and 0.5% respectively for the four runs. Results are shown in the following sections.

VI. Results

A. *Bandwidth, lossless*

Bandwidth results are shown in Figure 4. Note that the lines between the data points are to aid in visual grouping, not to imply a linear relationship. Bandwidth is shown as a percentage of the bandwidth required to send the data uncompressed and is equivalent to the compressed size of the data as a percentage of the uncompressed size. Collaboration between the streams made significant improvements in bandwidth usage for most of the algorithms. The CenceMe data was not highly correlated causing TinyPack-Collaborative to only improve upon the TinyPack-Init codes by a small fraction. In contrast, compression of the GATech Vehicular dataset benefited greatly from the TinyPack-C algorithm since the data contained a high degree of correlation between the streams at a single sensor.

If no correlation is detected at all in the data, then TinyPack-Collaborative and TinyPack-Init should function identically in terms of bandwidth although TinyPack-Collaborative would consume more energy.

B. *Bandwidth, lossy*

Figure 5 shows the results of the error tolerant version of our algorithm. As with the lossless case, the introduction of correlation between the sensed streams on the individual sensor node significantly reduced the amount of bandwidth usage needed to transmit the data. As expected, all the algorithms performed better as more error was allowed in the system. The effect of leveraging correlation between the streams was roughly equivalent to the lossless case. The datasets that had high degrees of correlation saw the most benefit.

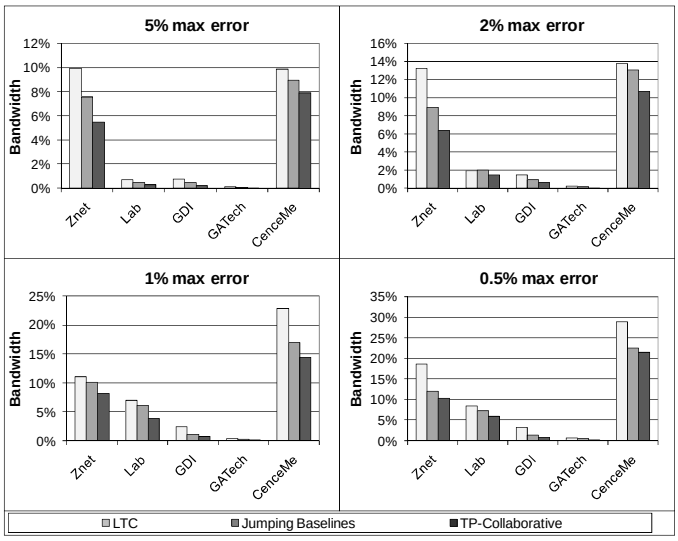

Figure 5 Bandwidth for lossy algorithms, all datasets

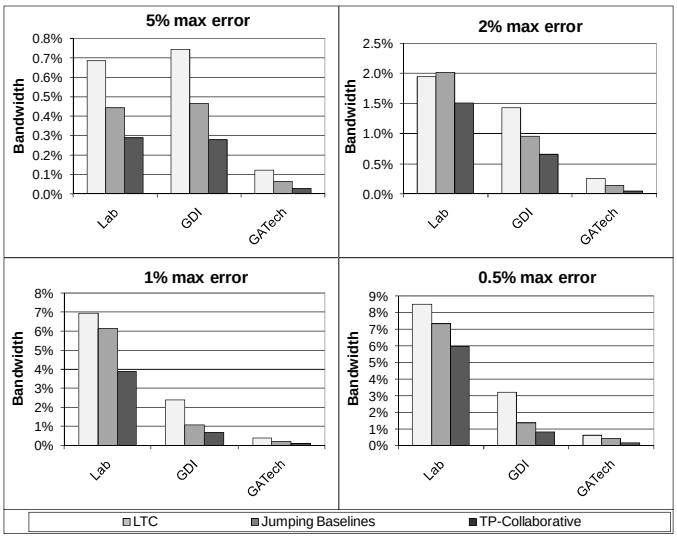

Figure 6 Bandwidth for lossy algorithms, selected datasets

The results vary greatly from one dataset to the next. This is due to the individual characteristics of the dataset. ZebraNet and CenceMe sensed data at a lower frequency than the others which decreases the benefits that can be gained by relying on temporal locality. The Lab, GDI, and GATech results are also shown in Figure 5 along with ZNet and CenceMe for comparison and are also shown in Figure 6 for greater clarity and readability.

As with the lossless case, the low degree of correlation in the CenceMe and ZNet dataset caused TinyPack-Collaborative to only perform slightly better than the other algorithms, while the GDI and GATech datasets were able to be consistently compressed to near or below half the size achieved by the Jumping Baseline algorithm.

While more tolerated error allowed for better compression in all cases, the relative compressed sizes for the different algorithms was roughly similar for all configured levels of tolerable error.

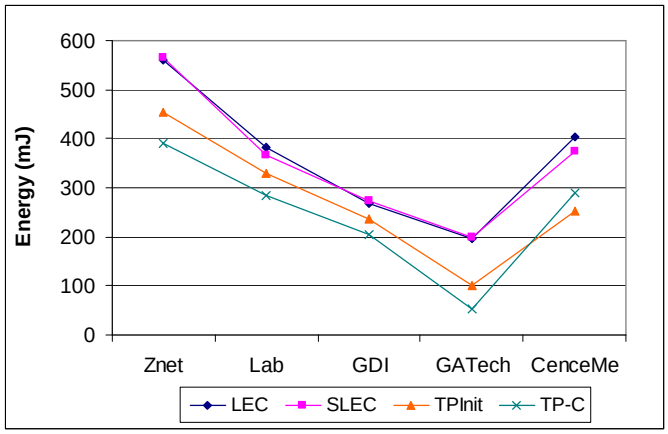

Figure 7 Energy consumption for lossless algorithms

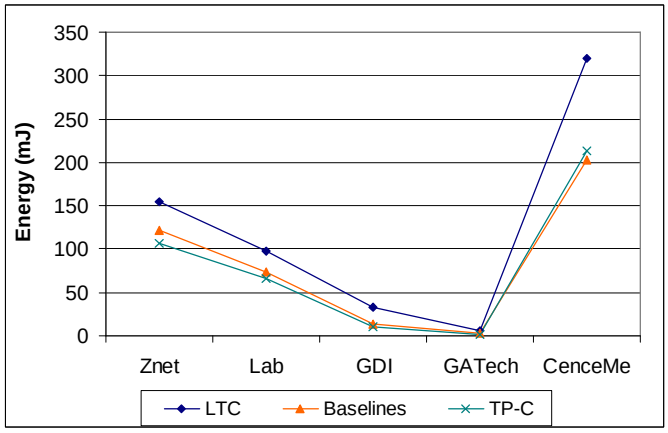

Figure 8 Energy consumption for lossy algorithms

C. *Energy*

The MicaZ motes simulated in PowerTOSSIM for measuring energy consumption have three different radio power settings that can be used requiring 11, 14, and 17.4 mA respectively. We selected the 11 mA radio for our experiments. Choosing a higher powered radio would make the results for energy consumption look almost identical to bandwidth since all the energy would be spent transmitting the data.

The results for the lossless case are shown in Figure 7. Since the bandwidth savings on CenceMe were not much greater for the TinyPack-C, the extra processor utilization was enough to cause it to require more energy than the jumping baseline method. The high number of streams in the GDI dataset caused a higher increase in the energy requirements for TinyPack-C relative to the other datasets. Even using the low powered radios, the bandwidth savings are still enough to cause a lower energy profile for sensors running TinyPack-C over the other algorithms for most datasets.

The results for the lossy case are shown in Figure 8 based on the 1% maximum error configuration. The lower bandwidth requirements of the error tolerant algorithms cause the increased processor utilization to have a more significant impact on overall energy consumption; however, energy consumption for TinyPack-C was still close to or better than the other algorithms for all the datasets studied.

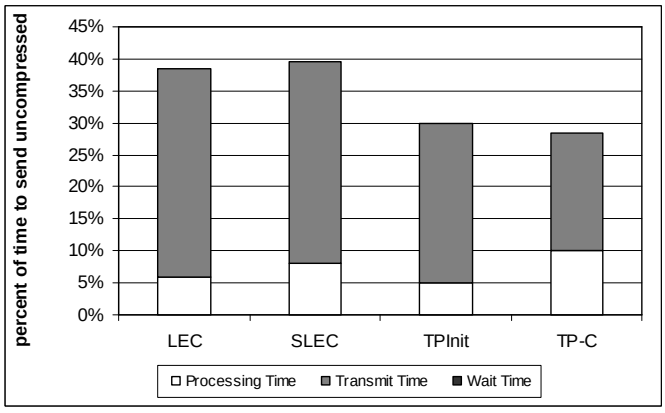

Figure 9 Latency for lossless algorithms

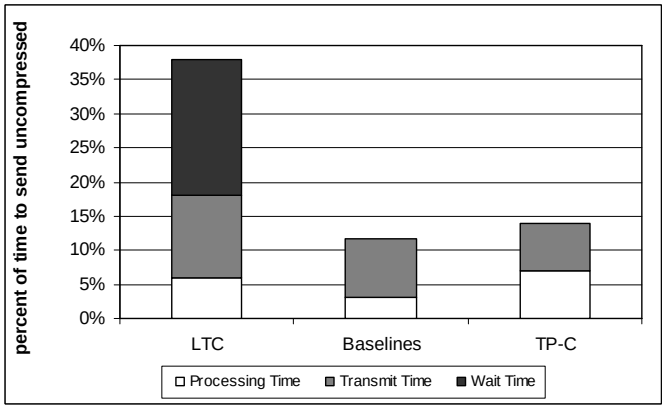

Figure 10 Latency for lossy algorithms

D. *Latency*

Latency results are shown for the lossless methods in Figure 9 and for lossy in Figure 10. Latency is shown as a percentage of the time that would be required to transmit the data uncompressed. Results are shown as the average across all the datasets including the processing, transmission, and wait time used by the algorithms.

As with energy, the higher processor utilization for TinyPack-Collaborative caused an increase in latency compared to the lighter weight TinyPack-Init and jumping baseline methods; however, in a multi-hop environment, the average latency per hop decreases with each hop and approaches the sum of the transmit time and the wait time as shown in Figure 11.

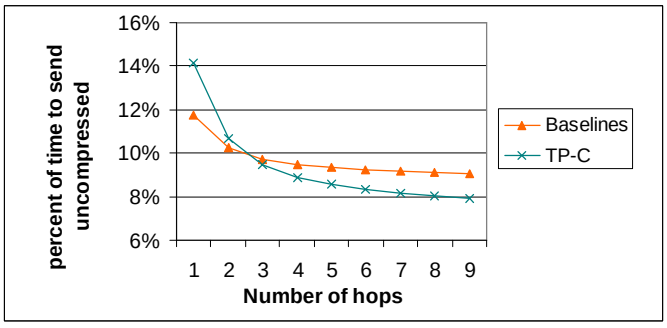

Figure 11 Latency for multi-hop environment

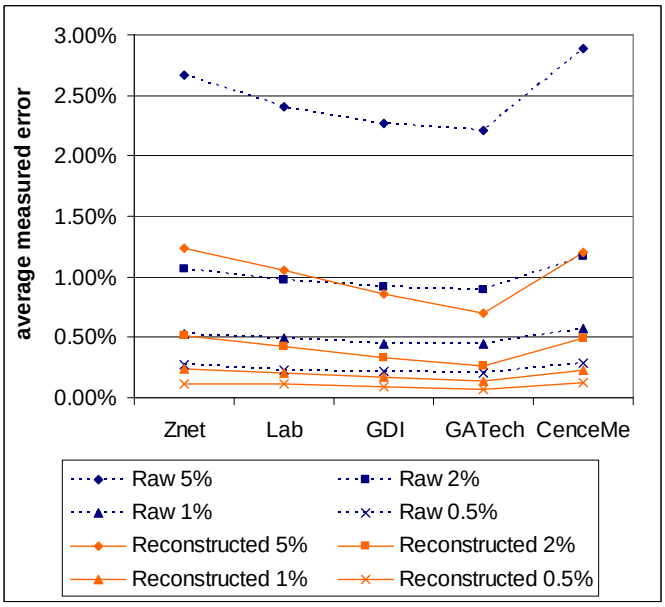

Figure 12 Average total error for raw baseline and reconstructed

VII. Error Analysis

The step function used to approximate the stream in the lossy case can be reconstructed into a series of line segments as done for the jumping baselines in [11]. This can reduce the total measured error in the data. The points at which new baselines were selected are used as the endpoints of the line segments.

Since the algorithm tracks whether the data was trending up, trending down, or peaking, this information can be used to better approximate the end points. If the data was trending up or down, then the line segment endpoint is selected as the average of the previous and current baselines. If the data is peaking (last jump was up, current jump was down or vice versa), then the previous baseline value serves as the endpoint.

Figure 12 shows the total error for both the raw baseline step function and for the reconstructed streams for each of the four configured maximum error percentages. Total error for the step functions is shown as dotted lines. The total error after reconstructing the streams as sequences of line segments are shown as solid lines. Data points for both raw and reconstructed for the same maximum error are shown with the same shape in the figure. Again, the lines between data points are to aid in visual grouping, not to imply a linear relationship.

Raw baseline step function total error was typically around one half of the maximum tolerable error. This is expected since the candidate baselines are integer multiples of the maximum tolerated error. The total error for the reconstructed streams ranged from around one quarter to one sixth of the maximum tolerable error. The more the data in a stream approximates a straight line over a short interval, the more accurate the reconstruction.

Experiments were also conducted using b-spline interpolation as a curve fitting technique, but the results were almost identical to the linear approximation and were much more computationally intense.

VIII. Aggregation of Compressed Values

As detailed previously, TinyPack-Collaborative, for both lossless and lossy compression, transmits values as the delta over some previous value or baseline function encoded using the TinyPack-Init codes. Some mathematical operations and aggregation can be performed on these encoded deltas without the need to first decode the data.

For instance, in an ad-hoc network, if an intermediate node between the sensor publishing the data and the base station begins forwarding data without seeing the initial baseline value, it can still perform aggregations on the data which the base station can apply to the baseline.

A. *Adding encoded values*

Adding two encoded deltas can be done without converting the value to a standard encoded integer. The codes contain a prefix, a suffix and a sign bit. In the case of two positive or two negative numbers, the two suffixes with their prefix bits prepended can be added in simple binary, if the high prefix bit overflows (is set to 0), then the prefix length is incremented by one and the sign bit remains unchanged. In the case of a positive and negative number, the negative number is expressed in 2's complement. The two numbers are added as before and the prefix length is reduced by the number of leading zeros in the sum.

B. *Dropping packets*

If a sensor network is being overloaded such that a sensor needs to conserve additional bandwidth, one common method for quick bandwidth savings is to drop a packet. In a compressed stream, simply dropping a packet causes the decoding process to produce incorrect results; however, delta compressors such as TinyPack-Collaborative can drop packets without invalidating the data as long as the delta values of all the dropped packets are summed into the next transmitted packet. For example, if a sensor received the values 5, 7, 12 9 10 and transmitted them as +5, +2, +5, -3, +1 and needed to drop every other packet, it could send +5, +7, -2 and the sink would decode them as 5, 12, 10. Any intermediate nodes need not know the baseline on which the first packet is based.

C. *Minimum and maximum*

Maintaining the maximum of a portion of a stream can be done without knowing the baseline by maintaining the current max delta and offset from the max delta by summing the delta values. For example, consider a sensor in an ad hoc network that samples the following values: 15, 13, 10, 12, 17, 13. The 15 is transmitted to the base station through one intermediate node and the remaining values through another node. The new intermediate node first sees the -2 and maintains the max as shown in Table V. Minimum can be maintained equivalently.

Table V Max Delta Example

sensed value	sent delta	current max delta	offset from max	actual max (delta+15)
15	--	--	--	15
13	-2	0	2	15
10	-3	0	5	15
12	+2	0	3	15
17	+5	+2	0	17
13	-4	+2	4	15

D. Average

Maintaining an average of a portion of a stream can be done without knowing the baseline as long as the count of samples included in the average is transmitted. The intermediate sensor maintains the current offset by keeping a running sum of the delta values. The sensor then maintains a sum of those offsets. Dividing that sum of offsets by the count gives the average delta value which can be added by the base station to the known baseline value to obtain the overall average. For example, consider a sensor that samples the following values: 10, 13, 17, 14, 8, 7, 15. Again, the intermediate node starts receiving and forwarding the data in the middle of the stream starting with the 13. This process is shown in Table VI.

Table VI AVERAGE DELTA EXAMPLE

sensed value	sent delta	sum of deltas	sum of sums	count	avg delta	actual avg (delta+10)
10	--	--	--	0		--
13	+3	+3	+3	1	3	13
17	+4	+7	+10	2	5	15
14	-3	+4	+14	3	4.67	14.67
8	-6	-2	+12	4	3	13
7	-1	-3	+9	5	1.8	11.8
13	+6	+3	+12	6	2	12

IX. CONCLUSIONS AND FUTURE WORK

TinyPack-Collaborative compression performed well compared to related methods in terms of bandwidth usage, energy requirements, and end-to-end latency. Collaboration between the data streams improved the compression performance in all experiments compared to compression without inter-stream collaboration. While collaboration between the same streams on different sensor nodes has been shown to be effective in increasing compression gains in other published works, collaboration between streams on the same sensor node can also be used to achieve greater compression leading to longer deployments, more data collection, fewer collisions, and faster response times for a wide variety of wireless sensor applications.

While the rolling least squares regression used here was shown to be effective, other more sophisticated methods such as Kalman Filters [22] or Principal Component Analysis [23] could be potentially improve the accuracy of the baseline correlation functions. It would also be useful to study the effect of node failures on compression and error calculations.

REFERENCES

[1] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, “Wireless sensor networks for habitat monitoring,” In WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications. New York, NY, USA: ACM, 2002, pp. 88-97.

[2] P. Bodik, W. Hong, C. Guestrin, S. Madden, M. Paskin, and R. Thibaux. Intel Berkeley Labs. 2004

[3] P. Zhang, C. M. Sadler, S. A. Lyon, and M. Martonosi. “Hardware Design Experiences in ZebraNet.” In Proc. of the ACM Conf. on Embedded Networked Sensor Systems (SenSys), 2004.

[4] Sadler C. and Martonosi M. "Data Compression Algorithms for Energy-Constrained Devices in Delay Tolerant Networks," In Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys), 2006.

[5] F. Marcelloni and M. Vecchio, "An Efficient Lossless Compression Algorithm for Tiny Nodes of Monitoring Wireless Sensor Networks," Computer Journal, vol. 52, no. 8, pp. 969–987, 2009.

[6] T. Szalapski and S. Madria, "On Compressing Data in Wireless Sensor Networks For Energy Efficiency and Real Time Delivery," In Distributed and Parallel Databases. June 2013, Volume 31, Issue 2, pp 151-182.

[7] A. Rooshenas, H.R Rabiee, A. Movaghar, M.Y. Naderi. "Reducing the data transmission in Wireless Sensor Networks using the Principal Component Analysis." Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP) 133-138, 7-10 Dec. 2010

[8] R. Masiero, G. Quer, M. Rossi. M. Zorzi.. "A Bayesian analysis of compressive sensing data recovery in wireless sensor networks." In Ultra Modern Telecommunications & Workshops, 2009. (ICUMT'09). 1-6. 2009.

[9] S. Gandhi, S. Nath, S. Suri, and J. Liu. "GAMPS: Compressing Multi Sensor Data by Grouping and Amplitude Scaling," In Proceedings of the 35th SIGMOD international Conference on Management of Data, New York, NY, 771-784. 2009.

[10] A. Ali, A. Khelil, P. Szczytowski, and N. Suri. "An adaptive and composite spatio-temporal data compression approach for wireless sensor networks." In Proceedings of the 14th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems (MSWiM '11). ACM, New York, NY, USA, 67-76.

[11] T. Szalapski and S. Madria, " Energy Efficient Distributed Grouping and Scaling for Real-Time Data Compression in Sensor Networks." In communication.

[12] Y. Liang, Y. Li. "An Efficient and Robust Data Compression Algorithm in Wireless Sensor Networks." Communications Letters, IEEE, vol.18. 439-442. March 2014

[13] T. Schoellhammer, B. Greenstein, E. Osterweil, M. Wimbrow, D. Estrin. "Lightweight temporal compression of microclimate datasets [wireless sensor networks]." 29th Annual IEEE International Conference on Local Computer Networks. 16-18 Nov. 2004.

[14] A. Vahidi, A. Stefanopoulou, and H. Peng. "Recursive least squares with forgetting for online estimation of vehicle mass and road grade: theory and experiments." Vehicle System Dynamics 43. pp. 31-55. 2005.

[15] M. Salgado, G. C. Goodwin, and R. H. Middleton. "Modified least squares algorithm incorporating exponential resetting and forgetting."International Journal of Control 47, no. 2 pp. 477-491. 1988.

[16] R. M. Fujimoto, R. Guensler, M. P. Hunter, H. Wu, M. Palekar, J. Lee, and J. Ko. "CRAWDAD dataset gatech/vehicular. v. 2006-03-15. Downloaded from http://crawdad.org/gatech/vehicular. Mar 2006.

[17] E. Miluzzo, N. D. Lane, K. Fodor, R. Peterson, H. Lu, M. Musolesi, S. B. Eisenman, X. Zheng, and A. T. Campbell. "Sensing meets mobile social networks: the design, implementation and evaluation of the cenceme application." In Proceedings of the 6th ACM conference on Embedded network sensor systems, pp. 337-350. ACM, 2008.

[18] P. Levis, N. Lee, M. Welsh, and D. Culler. "TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys) 2003.

[19] Crossbow Technology, Inc. MicaZ Datasheet. http://www.xbow.com/, 2010.

[20] V. Shnayder, M. Hempstead, B. Chen, G. W. Allen, and M. Welsh, "Simulating the Power Consumption of Large-Scale Sensor Network Applications," In Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys), 2004.

[21] Willow Technologies. http://willow.co.uk/TelosB_Datasheet.pdf, 2013.

[22] R. Olfati-Saber. "Distributed Kalman filtering for sensor networks," In Decision and Control, 2007 46th IEEE Conference on. Dec. 2007.

[23] A. Rooshenas, H. R. Rabiee, A. Movaghar, and M. Y. Naderi. "Reducing the data transmission in wireless sensor networks using the principal component analysis." In Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2010 Sixth International Conference on, pp. 133-138. IEEE, 2010.