Energy Efficient Distributed Grouping and Scaling for Real-Time Data Compression in 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 sensors in the network. Most of these algorithms require collecting a great deal of data before compressing which introduces an increase in latency that cannot be tolerated in real-time systems. We propose a distributed method for collaborative compression of correlated sensor data. The compression can be lossless or lossy with a parameter for maximum tolerable error. Error rate can be adjusted dynamically to increase compression under heavy load. Performance evaluations show comparable compression ratios to centralized methods and a decrease in latency and network bandwidth compared to some recent approaches.**

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

I. INTRODUCTION

Many real-time systems incorporate wireless sensors into their infrastructure. For example, some airplanes and automobiles use sensors to monitor the health of different physical components in the system, security systems use sensors to monitor boundaries and secure areas, and armies use sensors to track troops and targets. It is well known that wireless sensor networks possess significant limitations in processing, storage, bandwidth, and power. In addition, with the emergence of collaborative on-demand sensor applications [19], a need exists for efficient collaborative data algorithms that do not require delays in processing or communication while still reducing memory and energy requirements.

Data compression has existed since the early days of computers. Many new compression schemes for wireless sensor networks have been proposed. Many emphasize low energy profile [8][9] to function in the constrained wireless environment. Others exploit the physical layout of the sensors [2][3], or the spatio-temporal correlation often present in the data to achieve better compression. GAMPS [4] effectively uses spatio-temporal correlation by grouping correlated sensors and using amplitude scaling to relate the streams of values from the correlated sensors, but is centralized and requires collecting all of the data before compression. The distributed ASTC approach [7] performs the compression in-network by building and merging clusters and cliques of related sensors. It gives good compression ratios, but generates additional peer-to-peer communication and heavier energy usage from the increased processing.

We propose a distributed collaborative method designed for real-time sensor networks such as those used in the sensor cloud [19]. Correlated sensors form groups and use amplitude scaling on their signals to express their sensed values in terms of other sensors in the group. The grouping and scaling is done in a distributed fashion in real time. This is similar to the method used in GAMPS [4] which employs a centralized algorithm on the data after it has all been collected; however, GAMPS provides no reduction in bandwidth or energy use on the sensors and is not designed for real-time systems. Ours is a distributed approach which uses very lightweight algorithms that achieve high compression ratios to minimize bandwidth, latency, and energy usage.

If some loss in the accuracy of the data is tolerable, then the potential for compression increases greatly even for small loss. In our work here, we include a parameter for the maximum tolerable error for a single sensed value. For sensors with multiple inputs, the parameter can be set globally for all signals or individually for different error tolerance for each type of sensed value. Setting any max error to 0% naturally achieves lossless compression. We provide in-depth analysis and discussion of different methods for measuring error and compare the compressibility and actual error for variations methods of utilizing the error tolerance.

We then compare the results of our approach to the existing spatio-temporal existing methods such as GAMPS [4] and ASTC [7]. We also compare our method to the single sensor TinyPack [10] and LEC [9] methods and compare our prediction methodology with PREMON [6] and a sensor network adaptation of Kalman Filters [5]. Experiment and simulation results show significant reduction in bandwidth, latency, and energy consumption compared to the other methods.

In summary, this paper makes the following contributions:

- Novel adaptive algorithms for lossy collaborative compression in sensor networks with tunable maximum loss
- Discussion and analysis of how to select and handle tolerable loss in the data
- An ultra low-weight prediction mechanism
- An analysis of several methods of grouping and clustering
- Novel and effective error recovery techniques

978-1-4799-7575-4/14/$31.00 ©2014 IEEE

II. Related Work

A. GAMPS

A lossy multi-stream compressor is proposed in [4]. GAMPS compresses the data from multiple sensors that sense correlated data using mathematical techniques to groups the sensors that have highest correlation to each other. One sensor in each group is selected as the baseline and the rest of the sensors in the group report the difference in their sensed values from the baseline. The values are rounded based on a threshold parameter to achieve compressed sizes under 1% of the original size.

For a single sensor, the series of values is scanned until the difference between the maximum and minimum exceeds twice the error threshold. The entire sequence (excluding the last one, which caused the excess difference) is approximated as the average of the maximum and minimum. In this way the approximation never differs from the original by more than the error threshold. In order to keep the time windows consistent across all sensors in a group, the time slices are all reset when any sensor requires it.A baseline sensor exists in each group. Linear regression models are used to find the closest linear function that maps each sensor to the baseline. Again, if the error exceeds the threshold a new function is found.

The actual grouping is dependent on the above processes. An initial time window is set and the groups are set for each time window using a heuristic solution to the Facility Location problem. Initially all the sensors are in one group. Then a base sensor is chosen at random and sensors are added to its group as long as the cost of adding them is less than the cost of starting a new group. After the groups are set for each time window, the time windows are tested to see if halving or doubling will increase the compressibility of the data.

This method is very effective but requires full centralized knowledge of all the data before compression is possible at all. Therefore, compression cannot be done in-network so there are no savings in bandwidth or latency on the sensor nodes.

B. ASTC

In [7], a distributed, lossy, spatio-temporal approach is introduced. One-hop clusters comprised of correlated sensors are formed based on previous sensed values. A select number of the sensors in a cluster are chosen to form a master cluster on which temporal correlation is used to form a model. This model is sent to neighboring clusters, which can merge with the original cluster forming larger clusters.

Individual nodes that do not remain correlated to their respective clusters are evicted. These evicted nodes then listen to their neighboring clusters and can either join an existing cluster or form a new cluster depending on whether or not any of the neighboring clusters accept them.

The algorithm is effective, but all the communication in building the clusters causes additional latency and bandwidth requirements.

C. PREMON

PREMON [6] uses an algorithm similar to that of MPEG and JPEG compression. Sensor correlation is computed as vectors to macro blocks which are used to build a model for the data. The sensors then only report deviations from the model. All the computation of the models is done in a centralized fashion at the sink and the models are transmitted back to the sensors. The model is periodically reconstructed and retransmitted to the sensor nodes. As with ASTC, the additional communication costs significantly reduce the gains from compression. Also, the building of the model is expensive in terms of processor utilization so energy efficiency is lower.

D. LEC and TinyPack

A number of very lightweight compression codes are introduced in [9] and [10]. LEC consists of a set of delta compression codes based on JPEG compression and applied to sensor nodes. A similar set of codes is derived in TinyPack that is more highly tuned to the temporal correlation observed in many real life datasets. These codes are used as the basis for the delta compression used in reporting the deltas from the baseline values in this work.

III. Background

A. Collaborative compression

Compression on a single sensor can often be achieved by exploiting temporal correlation in the data. In the single sensor TinyPack algorithms [10], each sensed value is compressed using the most recent previously sensed value as a baseline and expressing the value as a function of that baseline. In multi-sensor environments, neighboring sensors can be used as the baseline allowing for greater compression under the assumption that the values from the two sensors are correlated.

B. Spatial locality

Wireless sensor networks where multiple sensors are deployed over an area generally exhibit spatial locality (data from readings taken by sensors geographically near each other are correlated). Any type of data that changes in a continuous fashion across space will be temporally located such as temperature, humidity, location of tracked objects, light intensity, distance to a sensed event, etc. In fact, it can be demonstrated that any network deployed over a certain area will either generate spatially located data or random noise.

Consider an arbitrary sensor network sensing a set of values $\{v_1, v_2, \ldots, v_{2N}\}$ sensed at locations $\{x_1, x_2, \ldots, x_{2N}\}$ where N is an integer. Assume that the values are not correlated. Then placing sensors at locations $\{x_1, x_3, \ldots, x_{2N-1}\}$ and $\{x_2, x_4, \ldots, x_{2N}\}$ would yield completely different values. Thus, offsetting the sensor locations would generate entirely different data. Therefore, excluding applications that generate pure noise, we can assume that readings at nearby sensors will be correlated.

Note that this does not apply to applications in which the sensors are deployed individually on specific locations such as those placed on animals for location tracking. These applications do not have any guarantee of spatial locality (although they may still exhibit spatial locality) and were not included in this study.

C. Data sets

The datasets used for the experiments were pulled from a variety of publicly available, real world sensor deployments.

The GATech Vehicular dataset (GATech) [11] was obtained testing a vehicle-to-vehicle network which sensed values such as the location, speed, and altitude of the vehicles. The Great Duck Island (GDI) [12] experiment deployed sensor nodes in and around the burrows of Leach's Storm Petrels to monitor various types of temperature, humidity, barometric pressure, and solar radiation. The Intel Berkley Labs (Lab) [13] deployment measured temperature, humidity, light intensity, and voltage inside a university lab. The ZebraNet project (ZNet) [14] 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.

IV. Tolerable Error and Prediction

We consider a parameterized maximum tolerable error percentage E_{max}. Instead of reporting every value exactly as sensed, if a value deviates from some 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. Measuring error

A common method of measuring error, E, between a reported value, V_R, and the actual value V_A, is shown in Equation 1.

$$E = \frac{|V_A - V_R|}{|V_A|} \quad (1)$$

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. For this work, error was measured as absolute error over the full range of the data using Equation 2.

$$E = \frac{|V_A - V_R|}{V_{MAX} - V_{MIN}} \quad (2)$$

B. Baseline selection

Let D be the maximum value by which a particular sensed value can differ from the baseline in order to maintain an error percentage within the upper bound E_{max}. Any time a value differs from the baseline by more than D, a new baseline must be selected. The easiest approach would be to simply use the current sensed value as the new baseline; however, the data is much more compressible if the baseline is chosen as kD where k is an integer such that kD is within D of the current sensed value. Since it is known that the baseline will jump to a multiple of D, only the difference between the current k and the previous k needs to be transmitted.

V. Single Sensor Compression

A. Baseline compression

We extend the benefit of jumping baselines for compression by implementing a simple prediction mechanism. 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. We determine the state 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. The prediction only requires caching the previous value and the previous jump direction. The additional computation is also trivial. For example, Table I shows an example of a light sensor with a maximum error set at +/- 10 lux.

Table I Prediction example

Seq no	Sensed value	Last value	Last jump	This jump	Baseline
1	242	237	--	--	240
2	253	242	--	up	250
3	261	253	up	up	270
4	276	261	up	--	270
5	284	276	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 value p, max difference S,
       previous jump direction d
Output: Reported value r
    If |p – v| > S
        r := NearestBaselineTo(v)
        If v > p And d == UP
            r := r + S/2
        Else if v < p And d == DOWN
            r := r – S/2
        End If
        If v > p
            d := UP
        Else
            d := DOWN
        End If
        p := r
    Else
        r := p
    End If
```

B. Entropy results

The total amount of bytes needed to transmit a data stream can be measured by the entropy of the dataset. Assuming no additional prediction methods are used for a data stream, the

entropy of the data (as defined by Shannon in [1]) provides a measure of the minimum number of bits that would be required to transmit the data if some theoretical optimal compression was used. Thus, entropy is an effective means of calculating the total "compressibility" of a data stream.

We used entropy to measure the effectiveness of the jumping baseline compression and prediction and compared the results to other prediction methods. PREMON [6] is an MPEG based prediction algorithm designed specifically for sensor networks. Kalman Filters are also commonly used to predict data streams. We compared against a Kalman filtering scheme that has been adapted for sensor networks [5]. PREMON and Kalman filters perform sophisticated prediction, reducing the number of messages that need to be sent while the jumping baseline method can afford higher compressibility. We also included the simplistic approach of merely rounding the data to the nearest baseline since that gives a similar reduction in entropy.

PREMON and rounding were configured to use the same maximum tolerable error and the Kalman Filters (which are not bounded on error) were configured to have the same total calculated error as the jumping baseline method.

The entropy of the transformed data as a percentage of the original entropy for the same data is shown in Figure 1. Four experiments were performed on each dataset with four different values set for maximum tolerable error. As expected, Kalman filters and PREMON required fewer messages to be sent due to more accurate prediction, but since the size of the messages would need to be higher, the jumping baselines performed best in terms of overall entropy. Thus compression will be more effective using the jumping baselines over the other methods.

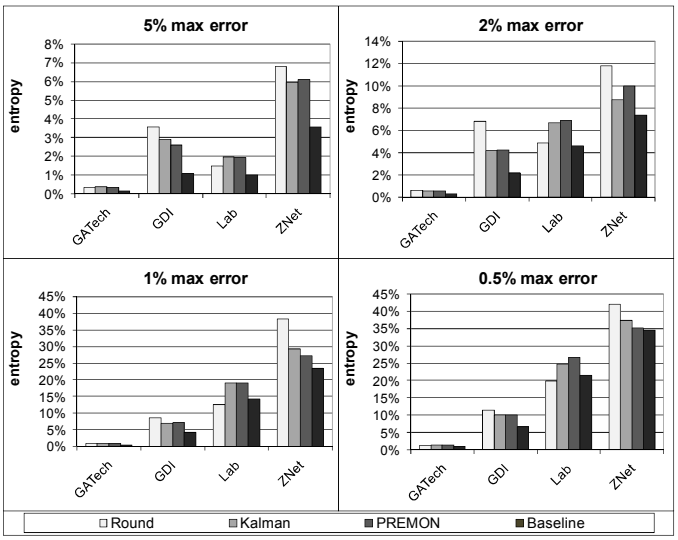

Figure 1 Entropy on varying max error for different prediction algorithms

Linear Predictive Coding was also investigated as a method for predicting node values. LPC has been specifically tuned for sensor networks [20] but only performs well when the data is audio streams and did not perform well on environmental datasets. Results were not included in this work.

VI. Collaborative Sensor Data Compression

A. Correlation

Collaboration between the sensors can then be used to further enhance the compression of the entire dataset. Correlated sensors can transmit the count of jumps in which their baselines differ. Each group of correlated nodes can be viewed as a tree. The root node of each tree is the sensor that sends its own data directly. The sensor chosen as the base sensor serves as a parent node in the correlation tree. Then the child node can report its values using its offset from the parent sensor's baseline as its baseline. Figure 2 shows an example of a seven node correlated group. Sensors S2, S5, and S6 are reporting their values using S3 as their baseline; sensors S4 and S1 are using S5 as a baseline; and sensor S7 is using S6 as its baseline. Note that this tree is just representative of correlation and is not related to transmission from sensing node to sink.

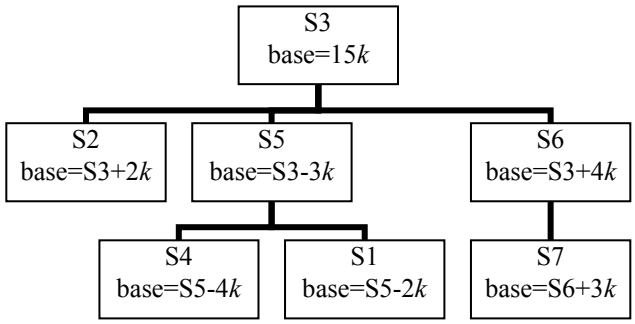

Figure 2 Collaboration hierarchy

The algorithm used is identical to Algorithm 1 except the total count of baseline jumps is reported as an offset of the other sensor instead of an absolute. For example, consider two light sensors where sensor S_2 is reporting its values based on sensor S_1. Assume again the maximum error is +/- 10 lux. Table II shows a sample data stream for the two sensors including the actual sensed values, the message sent, and the final reported value as interpreted at the sink.

Table II Collaboration example

Seq no	S_1 sensed	S_2 sensed	S_1 sent	S_2 sent	S_2 offset	S_1 final	S_2 final
1	237	259	+24	+2	+2	240	260
2	242	266			+2	240	260
3	253	271	+1		+2	250	270
4	261	278	+2	-1	+1	270	280
5	275	282			+1	270	280

At the first sensed values, the sensors have no baselines, so S_1 uses 0 as its baseline and S_2 uses S_1's initial value as its baseline. In the message at sequence number 3, S_2 would have needed to transmit a jump message if it were reporting its own values, but since S_1 reported a jump, S_2's interpreted value automatically jumped. Two noteworthy things happened at sequence number 4. The prediction detected the upward trend in S_1's data and selected the highest baseline within the tolerable error, and S_2 corrected its offset from S_1's baseline.

B. Codes

The codes used for transmitting the compressed baseline jumps for individual or correlated sensors are drawn from those used in [10]. An example set of codes for the delta values of -127 to +127 is shown in Table III. For example, a change value of +3 would be transmitted as 00101 and -3 would be transmitted as 00111. The pattern can continue for values as high as are needed. If the maximum value is known, the last level need not have a 1 at the end of the prefix.

Table III DELTA CODES

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

These codes can be used to both encode and decode very efficiently with minimal processor utilization. The value expressed by a code can be computed by the following equation where B is the number of 0 bits before a 1, S is the first bit of the suffix (sign bit) and k is the number represented by the remaining suffix bits interpreted as an integer:

$$(-1)^S \left(2^B + k\right) \tag{4}$$

For example, the value -14 would be represented by 0001 1 110 where prefix=0001 (thus $B = 3$ and $2^B = 8$), $S = 0$, and $k = 110 = 6$. So (-1)(8+6) = -14.

C. Messages

There are only two message types sent by the sensors: baseline jumps, and parent sensor changes (rebellions). Since these rebel messages are expected to be infrequent compared to the baseline jumps, it would be inefficient to assign an entire bit to distinguish between the message types. Instead a value is selected from the table to use as the indicator and all the other values are shifted down one. For our experiments, we used -15 since the frequency of rebel messages needed in our datasets was closest to the frequency of values in that range. So if a value started with 00011111, it is interpreted as a rebel message and the rest of the bits contain the new parent node ID. Then an actual -15 message would be encoded like -16 and so on. Node IDs are compressed by using the minimum number of bits needed for the total number of nodes. For example, if there were 33 to 64 nodes deployed, the IDs would use 6 bits.

Another small gain was obtained by shifting past known invalid values. For example, if a data stream is trending up, a jump of +1 would be invalid since it would be guaranteed jump by at least +2. For any jump in the direction a stream is trending, the value is encoded as if the absolute value was reduced by 1.

D. Grouping

Not all sensors in a network are necessarily correlated and the values from sensors that are correlated may not be equal. Distinct groups of sensors that exhibit higher correlation tend to emerge and the values at one sensor can often be more efficiently transmitted as a difference from another sensor's values. We compare using two very simple and lightweight grouping mechanisms: sink side and node side. Performance could potentially be increased by using more sophisticated grouping methods. This is left to be investigated in a future work.

The sink side approach assumes that the sink is not another sensor node and does not have the same energy and processing constraints. It also assumes that the sink can communicate back to the sensors. The node side method makes no assumptions.

In the sink side algorithm, the sink performs the facility location computations as done in [4] over a window of the recent data and reports back to the nodes the ideal parent node for that window. The weight of an edge between two nodes is defined as the amount of bits that would be required to transmit the data if one of the nodes was reporting its data based on the other node. The facility location algorithm is then performed to find the minimum number of bits required to send the data from all the nodes. Each "facility" selected by the algorithm serves as a base node for the group.

In the node side algorithm, the nodes maintain an array indexed by other node IDs with two entries. The first entry contains the current baseline jump distance from that node and the second entry contains the number of times the first entry has changed. Every time a node would need to send a jump message from its current parent, it finds the minimum jumps in the array and selects that node as its new parent. If two nodes select each other as the parent, the tie is broken by node ID and the node with the lower ID is selected as the parent.

If a node's parent node selects a parent, the node does not need to select a new parent. It merely calculates the value of its parent based on the reported value from the grandparent node. If the grandparent node is not in radio range however, the node will need to select a new parent.

VII. RESULTS

A. Bandwidth

Results for total bandwidth requirements are shown in Figure 3. Again, experiments were conducted using four different settings for the maximum tolerable error. We compared results between our baseline compression on single sensor, the GAMPS algorithm, ASTC, and our collaborative compression approach. The sink side algorithm performed almost identically to the node side algorithm but slightly worse due to the increased amount of messages sent and is not included in the graphs.

Bandwidth is shown as a percentage of the bandwidth required to send the data uncompressed. We assumed uncompressed data would be transmitted with the minimum number of bytes required to cover the observed range of possible values.

Collaborative baseline compression performed best in terms of required bandwidth compared to the other approaches for all data sets. The single sensor baseline compression performed almost as well for ZNet because the ZNet dataset did not

display as much correlation between nodes as was observed in the other datasets.

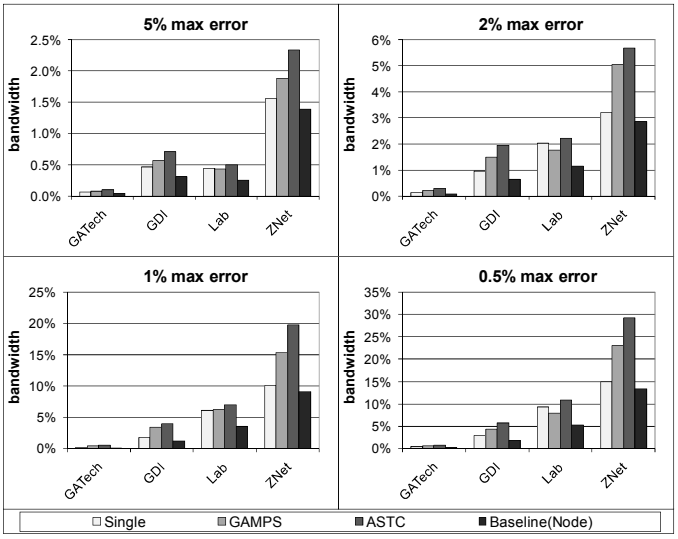

Figure 3 Bandwidth utilization on varying max error for different compression algorithms

B. Latency

Latency was measured in a network of TelosB motes [18] loaded with the data from each dataset and configured to send data to the sink based on the timestamps in the datasets. Figure 4 shows the latency results for the collaborative baseline compression and comparative methods. Results show time required to process the data, transmit the data, and any time required to wait to send the data. Results were aggregated across all the datasets.

Tolerable error only affected the transmit time. Results are shown for 5% max error for better clarity since at lower errors, the latency for processing would be difficult to see. The transmit time is a simple function of the compressed size of the data. At 5% max error, our collaborative baseline approach performed the best in terms of latency. As the tolerable error decreased, our single sensor baseline method had the least latency.

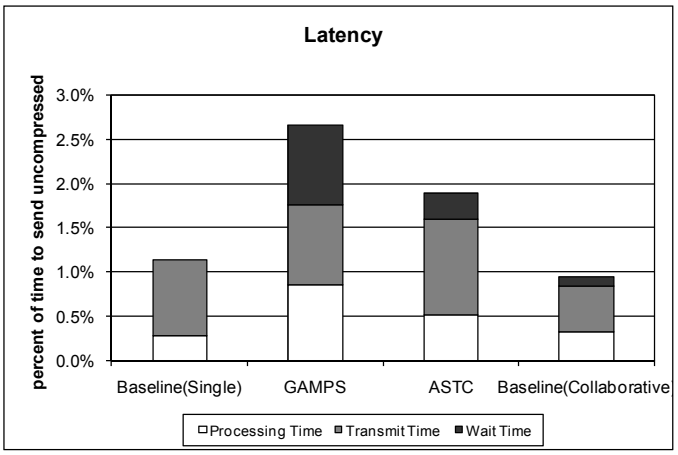

Figure 4 Total latency for single hop network for different compression algorithms

For comparison, GAMPS was modified to send data as soon as enough had been collected to perform the compression. ASTC incurred some wait time as the nodes communicated to build the prediction model. The nodes were not synchronized for all datasets, so for the jumping baseline, a correlated sensor reporting its value from a base sensor would occasionally need to delay sending its offset until the baseline sensor had sent its value.

Latency results shown are for a single hop network. As the number of hops increases, the total latency at each hop approaches the latency of the transmit time since no additional processing or wait time would be required. Since the collaborative baseline algorithm provided the best compression ratio, it performs better compared to the other algorithms as the number of hops between the sensing node and the final sink increases.

C. Energy Usage

A network of MicaZ motes [15] running TinyOS was simulated in TOSSIM [16]. Energy consumption was modeled using PowerTOSSIM [17] which provides a layer of energy usage tools on top of the sensor simulation tools provided in TOSSIM. Figure 5 shows the average energy per sensor required to compress the data for each of the algorithms. The energy required to transmit the data is a directly proportional to the compressed size of the data. Energy usage results for transmitting the data are not shown since they would be proportionally identical to the bandwidth results.

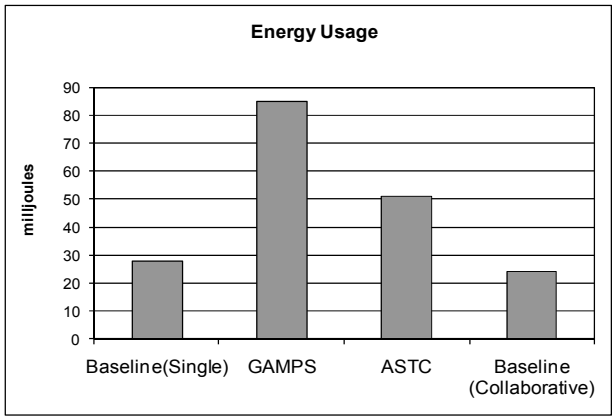

Figure 5 Energy usage due to processing for different compression algorithms

The MicaZ mote has three different radio power settings that require 11, 14, and 17.4 mA respectively while transmitting. The MicaZ processor uses 8 mA in active mode. The total energy required is dependent on the radio power setting. Since total energy consumption is based on current and time, the total energy results are proportional to the latency results for processing and transmission in Figure 5 except the transmission energy scales to 11/8, 14/8, or 17.4/8 of the transmission time based on the radio power used.

The simplicity of the jumping baseline approach gives it a much lower processing profile than the other methods. GAMPS was not designed to be energy efficient and as a result did not perform well. Baseline compression on a single sensor requires much less processing, but the savings from transmission energy

made the collaborative approach perform the best in terms of total energy consumption.

VIII. Error Recovery

We conducted some additional experiments on outlier detection and error recovery techniques to analyze their effect when coupled with the jumping baseline algorithm. These experiments were done over the individual streams of the four different data types sampled in the Intel Labs dataset (temperature, humidity, light intensity, and voltage).

A. Outliers

High fluctuation caused by errors in a stream can reduce the effectiveness of the jumping baseline algorithm. We defined an outlier detection window of size *W*. The readings in a window are considered outliers if the readings in the window differ from those immediately preceding and following the window by more than one baseline jump and the readings immediately preceding and following the window differ from each other by less than one baseline jump. In other words, if a sensed stream briefly reports a drastic change in value and then returns to the previous value, that change is likely to be an error and those readings are considered outliers. Manual inspection of the data revealed some clear outliers where a temperature reading or other value type would drop to 0 for a single sensed value and otherwise remain fairly constant.

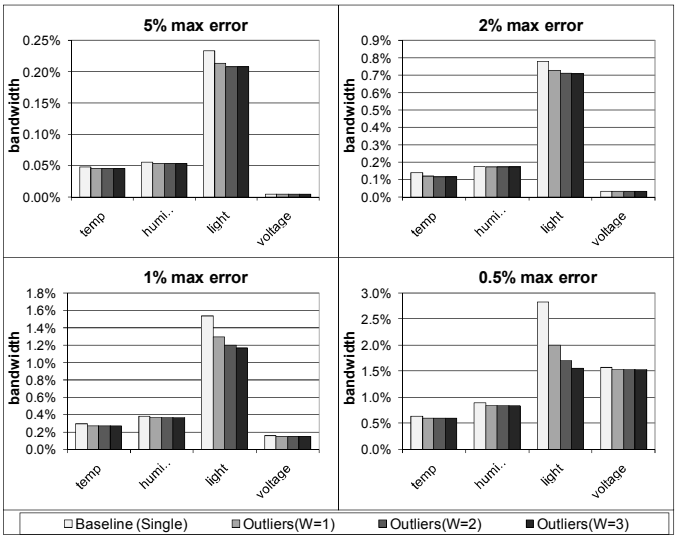

Figure 6 Bandwith savings with outlier detection

We performed simulations for window sizes of 1, 2, and 3. For window sizes greater than 1, any value that would be considered an outlier using a smaller window size is still considered an outlier. Results are shown in Figure 6. Outlier detection was most effective at the lower maximum tolerable error settings since reporting the outliers is more costly due the higher count of baseline jumps.

There were not many outliers detected in the dataset; however, on average, for a window size of 1, outliers comprised 0.11% of the data stream but required 7.4% of the bandwidth. Thus, detecting outliers in this way can significantly reduce the bandwidth required to send the data especially if the number of outliers is high.

Most of the outliers in the dataset were single values so increasing the window size above 1 did not cause more outliers to be found in all cases except for light intensity. The lights used in the experiment were fluorescent lights that produce a flickering affect. This flickering caused brief significant changes in the data stream that were calculated to be outliers. The question of whether or not such flickering should really be treated as outliers should be determined based on the goals of the individual experiment.

B. Signal reconstruction

The actual error present in the compressed stream can be reduced by using the compressed data to approximate the original data through curve smoothing techniques. Since the actual error is bounded by a maximum tolerable error *E*, the range of possible true values that produces the compressed stream is known. This can be used to aid the curve smoothing process and generate a more accurate reconstruction of the original data stream.

If the real data changes slowly and smoothly, this can provide a dramatic decrease in the actual error of the reported stream; however, if the data is highly varied within the bands, then attempts to reconstruct the original stream can actually add more error. The maximum added error is known, however, since it can be no more than twice the configured maximum tolerable error (assuming the reconstruction is designed to remain within *E* of the reported value from the compressed stream).

Due to the unique nature of the jumping baseline algorithm, when the baseline changes, the true value at that point can be accurately reconstructed. When data is trending up or down and the baseline jumps, the true value at the point of the jump will be nearly equal to the average of the two baselines. (If the sample interval was infinitely small, it would be exactly equal). When the data stream is peaking or oscillating (neither trending up nor down) the true value at a baseline jump can be accurately approximated by the value of the new baseline. Since the data trend is known, this can be used to design a very simple signal reconstruction algorithm that can greatly reduce the total error in the stream.

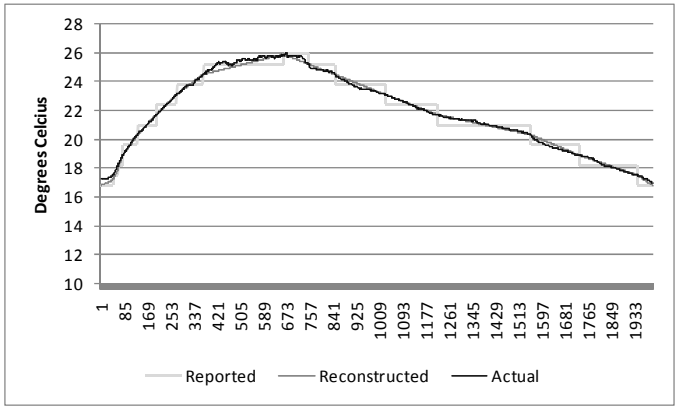

Figure 7 Reconstructed stream

The reconstructed stream is build by first approximating the values at the points where the baseline jumped. Then any curve fitting algorithm can be used to fit a curve to those points to

create the fully reconstructed stream. For testing, we simply approximate the curve by assuming the data between the points is linear. Figure 7 shows 2000 readings from a temperature sensor compressed using the jumping baseline algorithm comparing the actual stream of values with both the compressed stream reported to the sink and the reconstructed stream.

We computed the actual error both with and without signal reconstruction for different configured max tolerable errors over the entire dataset. Temperature, humidity, and light intensity all were very similar. Signal reconstruction reduced the measured average error to approximately 1/6 of the max tolerable error. Aggregated results are shown in Table IV. Voltage streams were not as continuous as the other three and signal reconstruction was not as effective. The actual error of the voltage streams after reconstruction was approximately 1/3 of the max tolerable error for each configured maximum used in the experiments. Voltage results are shown in Table V.

Table IV ERROR (TEMPERATURE, HUMIDITY, LIGHT)

Max tolerable error	Baseline error	Reconstructed error
5%	2.47%	0.832%
2%	0.964%	0.323%
1%	0.483%	0.167%
0.5%	0.239%	0.0815%

Table V ERROR (VOLTAGE)

Max tolerable error	Baseline error	Reconstructed error
5%	2.56%	1.38%
2%	1.06%	0.692%
1%	0.519%	0.387%
0.5%	0.252%	0.193%

The unique properties of the jumping baseline compression allow for a significant reduction in absolute error after a simple reconstruction algorithm.

IX. CONCLUSIONS AND FUTURE WORK

The jumping baseline method provides a very light weight collaborative compression scheme for wireless sensor networks. Energy and processing usage were well below those of existing algorithms while maintaining lower latency and requiring less bandwidth.

Data fusion and sequential processing of the data in a multi-hop sensor environment could be implemented using similar correlation algorithms to achieve even greater compression.

Compression could be improved even further in the future by taking advantage of correlations, not only between neighboring sensors, but also between different streams on the same sensor. For example, temperature and light were somewhat proportional in the dataset and were inversely proportional to humidity.

Since signal reconstruction could be done on the sink side, much more sophisticated algorithms could be used to fit a curve to the values approximated at the jump points.

REFERENCES

[1] C.E. Shannon, "A Mathematical Theory of Communication", Bell System Technical Journal, vol. 27, pp. 379–423, 623-656, October, 1948.

[2] T. Arici, B. Gedik, Y. Altunbasak, and L. Liu, "PINCO: a Pipelined In-Network Compression Scheme for Data Collection in Wireless Sensor Networks," In Proceedings of 12th International Conference on Computer Communications and Networks, October 2003.

[3] D. Petrovic, R. C. Shah, K. Ramchandran, J. Rabaey, "Data Funneling: Routing with Aggregation and Compression for Wireless Sensor Networks," In Proceedings of First IEEE International Workshop on Sensor Network Protocols and Applications, May 2003.

[4] 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.

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

[6] S. Goel and T. Imielinski. "Prediction-based monitoring in sensor networks: taking lessons from MPEG." In SIGCOMM Computer Communications Rev. 31, 5 (October 2001), 82-98.

[7] 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.

[8] 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.

[9] 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.

[10] T. Szalapski and S. Madria, "Real-Time Data Compression in Wireless Sensor Networks," In the 12th International Conference on Mobile Data Management, 2011.

[11] 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.

[12] 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.

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

[14] 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.

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

[16] 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.

[17] 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.

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

[19] Sanjay Madria, Vimal Kumar and Rashmi Dalvi, "Sensor Cloud: A Cloud of Virtual Sensors," IEEE Software, Nov 2013.

[20] Puthenpurayil, Sebastian, Ruirui Gu, and Shuvra S. Bhattacharyya. "Energy-aware data compression for wireless sensor networks." IEEE International Conference on Acoustics, Speech and Signal Processing Vol. 2. 2007.