Lightweight Temporal Compression of Microclimate Datasets

Tom Schoellhammer
University of California Los Angeles
Los Angeles, CA
tom@cs.ucla.edu

Ben Greenstein
University of California Los Angeles
Los Angeles, CA
ben@cs.ucla.edu

Eric Osterweil
University of California Los Angeles
Los Angeles, CA
eoster@cs.ucla.edu

Mike Wimbrow
University of California
Idyllwild, CA
mike.wimbrow@jamesreserve.edu

Deborah Estrin
University of California Los Angeles
Los Angeles, CA
destrin@cs.ucla.edu

Abstract

Since the inception of sensor networks, in-network processing has been touted as the enabling technology for long-lived deployments. Radio communication is the overriding consumer of energy in such networks. Therefore, data reduction before transmission, either by compression or feature extraction, will directly and significantly increase network lifetime.

In many cases, it is premature to begin implementing feature extraction techniques. Users do not yet understand in what forms interesting data will appear and consequently can't risk automatically discarding what they presume to be uninteresting. Moreover, computer scientists are only beginning to develop algorithms to collect spatially distributed features in situ.

Even for the many application where all of the data must be transported out of the network, data may be compressed before transport, so long as the chosen compression technique can operate under the stringent resource constraints of low-power nodes and induces only tolerable errors. This paper evaluates a simple temporal compression scheme designed specifically to be used by mica motes for the compaction of microclimate data. The algorithm makes use of the observation that over a small enough window of time, samples of microclimate data are linear. It finds such windows and generates a series of line segments that accurately represent the data. It compresses data up to 20-to-1 while introducing error on the order of the sensor hardware's specified margin of error. Furthermore it is simple, consumes little CPU and requires very little storage when compared to other compression techniques. This paper describes the technique and results using a dataset from a one-year microclimate deployment.

1. Introduction

Computer scientists are succeeding in producing massively distributed, cheap, untethered, and extraordinarily resource-constrained sensor networks that sample the environment at spatial resolutions never before possible and deliver rich environmental data to human users[1, 2].

Of the enabling technologies that are under development, in-network processing has long been touted as the key to running long-lived applications on power-constrained devices. Raw data is compressed or transformed *in-network* into tighter representations of the information sampled, and hence, fewer bits are transmitted by the relatively power-hungry radios of sensor nodes.

When it is known a priori which features in a dataset will be of interest, in-network processing algorithms can be tuned to focus on such features and disregard extraneous information. When it is not known what will be of interest, data reduction can be achieved using either lossless compression techniques that reduce data representations slightly, but preserve all data precisely, or by lossy compression techniques that repre-

sent data approximately and thus can reduce data arbitrarily.

Scientists, in practice, do not yet know a priori which features will be interesting because they have never before sampled their domains so densely. Therefore, they must collect as much of the original raw dataset with as little information loss as possible. Initially and periodically, scientists want, without loss, all of the raw data that a sensor network can produce.

Sending every bit of data up a routing tree and through a gateway to a query source is not scalable as the number of nodes increases, either in terms of the average or bottleneck energy consumption due to communication. The depth of such a routing tree is proportional to the diameter of the network as measured in communication hops. Thus, for n nodes that each generate a single packet of sensor data, the average node transmits $\sqrt{n}$ packets; the bottleneck must receive n packets. Likewise, lossless raw data collection isn't scalable because of bandwidth constraints; a network that samples data every t seconds cannot expect its bottleneck to receive n packets every t seconds unless n/t is small.

This paper proposes a new and computationally simple compression technique developed for the context of habitat monitoring that obtains up to a 20-to-1 reduction in the amount of environmental data that needs to be transmitted on certain data sets. Our technique, called *Lightweight Temporal Compression (LTC)*, introduces a small amount of error into each reading, bounded by a control knob. The larger the bound on this error, the greater the savings from compression. We have achieved significant savings even when the introduced error is less than the margin of error specified on a sensor's data sheet.

LTC has been implemented and integrated into the query engine component of the Extensible Sensing System (ESS) application being developed for James Reserve.

In section 2 we survey other work relevant to our scheme. LTC was motivated by the lifetime requirements of the ESS deployment. We discuss this deployment scenario and associated hardware and software infrastructure in section 3. In section 4 we describe our algorithm. Section 5 presents an analysis of our experimental results. We conclude in section 6.

2. Related Work

Three general techniques can be applied to reduce datasets in-network: extraction of features and statistics by in-network aggregation, compression by spatial correlation, and compression by temporal correlation. Our algorithm falls into the last category. The other two categories are complementary to our work.

2.1. Aggregation

Perhaps the first major work in sensor network data processing is Directed Diffusion [3]. Diffusion sought to provide a framework for in-network processing. The deployment for which we implemented our scheme uses a variant of Diffusion for routing.

TinyDB [4, 5, 6] is an in-network database that streams data and statistics on that data from sources to sink. For example, the average temperature may be computed hop-by-hop in a routing tree by recursively maintaining a sum and count for each subtree. TinyDB provides an elegant and simple to use SQL-like interface, supports a range of queries, and has been rigorously tested over a wide range of deployment conditions.

2.2. Spatial Correlation

Wavelet compression has been proposed for use in sensor networks by Ganesan et al. in Dimensions [7, 8, 9]. Wavelet compression is a good match for sensor networks. One may use wavelets to view data at multiple spatial scales and extract important features such as abrupt changes.

When data can be modeled, many statistical techniques may be applied to reduce the data transmitted. In a technique based on kernel linear regression, nodes communicate constraints on model parameters, drastically reducing the communication required [10]. The model takes the form of a weighted sum of local basis functions. Another technique uses model-based compression for shortest paths routing tables [11].

Likewise, DISCUS [12, 13] uses distributed source coding (compression of multiple correlated sensor outputs that do not communicate with each other) and joint decoding at a base station. It considers lossless (Slepian-Wolf) and lossy (Wyner-Ziv) source coding. DISCUS suffers the same drawback as all model-based techniques; it is hard to come up with a joint probability density function in sensor networks, particularly when there is little room for training about the deployment environment.

2.3. Temporal Compression

Two popular and successful lossless text compression schemes, namely BZIP2 based on Burrow Wheelers Transform (BWT) [14] and GZIP based on Lempel-Ziv (LZ) [15] family were designed to be used by PC-grade devices for the compression of data files. These

techniques can be directly applied to time-series data. We compare our scheme to LZW to show its performance relative to these schemes.

Wavelet compression is a high performance, lossy compression scheme that may be used in the temporal domain. Since our compression scheme is inherently lossy we also compare it to wavelet compression.

Other sensor network schemes include PINCO [16] for data compression, compression for time delay estimation[17], and energy-efficient communication via compression [18].

3. Target Deployment Scenario

The Extensible Sensing System (ESS) is under development for a deployment at James Reserve [19] in the San Jacinto Mountains. This is a long-term project designed to provide spatially dense environmental, physiological and ecological information to scientists. It has also given computer scientists experience designing and densely deploying long-lived, wireless sensor networks. ESS focuses on monitoring microclimate and other physical characteristics of the plant and animal habitats, encompassing below ground root observation and sensing, water movement through soils near roots, hydration status of mosses and lichens, and growth phenology of herbaceous, shrub and canopy species.

The ESS architecture consists of motes connected to weather sensing boards that communicate via low-power ChipCon radios with Stargate [20] microservers that in turn are connected to an Oracle back-end via 802.11. Since power and bandwidth are limited on motes, but not on the microserver/database backchannel, motes dynamically select the closest microserver when forwarding data. Doing so reduces the aggregate communication of the network, and, hence, prolongs network lifetime.

The software architecture consists of three components: a sampler, a routing and in-network processing framework, and a query processor.

3.1. Sampler

The sampler coordinates the sampling requests from the query engine and tasks the appropriate sensor drivers to collect data. It is designed for microclimate deployments, and consequently may be used to sample external temperature (air, soil, water), humidity, soil moisture, leaf wetness, wind speed and direction, pressure, radiation (including photosynthetically reactive band sampling or PAR), and to detect motion.

3.2. Routing and In-Network Processing

ESS uses Diffusion's one-phase-pull [21] protocol for transport across motes to and from microservers. Microservers disseminate information such as queries and control beacons in Diffusion interests which are flooded. Data is returned to the roots of a forest of routing trees via Diffusion gradient forwarding.

3.3. Query Processing

The query processor was designed to provide ecologists with a means of acquiring the data that they are interested in. It is tailored specifically to the needs of scientists at James Reserve. Five query types have been identified by scientists and these are currently supported by the query processor. As new query types are developed, the query processor will be extended to support them.

- A *single shot* query requests data from a specified set of sensors be returned once.
- A *periodic* query requests periodic sampling from a set of samplers. One period is associated with the set.
- A *conditional periodic* query threshold tests the value of one sensing modality, and samples a named set of modalities when the test passes.
- Some sensors report when events occur. For example a digital sensor connected to a rain bucket that tips over when full signals when it has done so; likewise a wind gauge signals an event each time its wind blades rotate. A *triggered* query samples a number of sensing modalities when such an event is received. E.g., scientists find it useful to know the temperature and humidity when the rain bucket tips.
- Finally, when an event occurs, a *triggered aggregate* query takes a time window of values from each of a desired set of sensing modalities and return statistics, such as the maximum or average value, on those windows.

Since the ESS deployment will cover a large area, nodes must be untethered and largely unmanned. It simply isn't feasible to continuously replace mote batteries. In order to increase node lifetimes, we have added lightweight temporal compression to the query processor.

3.4. Compression Alleviates Constraints of Scale

The state of the art for data collection in sensor networks is to form a multi-hop routing tree rooted at a network gateway. Routing trees are simple to construct and require little state.

It is necessary to employ a compression scheme when the aggregate data of the network is produced at a rate in excess of the bandwidth near the root of the tree. Otherwise, all data cannot be transmitted out of the network. Compression does not fix the lack of scalability of routing trees in this regard, but by reducing aggregate traffic by a constant factor, can alleviate the problem significantly. Likewise, when applications store data locally, compression can be used to reduce the data representation, so that more data can be stored.

Moreover, compression increases network lifetime. Since the radio is the dominant consumer of energy in a sensor node [22, 23], savings due to compression directly translate into lifetime extension for network nodes. I.e., a network that employs a codec that achieves 20-to-1 compression will live approximately twenty times longer than one that transmits raw data.

4. Algorithm

In this section we first provide a characterization of sensor noise and environmental data which motivates the design of LTC. We then go onto to show why existing compression techniques are not feasible for motes, and how LTC is both feasible for motes (in terms of resource requirements) and performs well.

4.1. Sensor Data is Noisy

Even when a sensor is sampling an unchanging phenomenon the sensor will produce a range of readings due to *noise*. Consequently, sensor manufacturers tend not only to specify a sensor's *operating range* but also specify a sensor's *accuracy*.

The data sheets for most environmental sensors characterize the hardware as producing accurate values within a defined *margin* of error. For some examples, the temperature sensor used in the ESS deployment is listed as having a 0.5°C margin and the relative humidity sensor is listed as having a 2% margin. It is unusual for datasheets to include a probability distribution that further characterizes this error. In other words, when a sensor value is taken, it is known with great certainty that the actual value of the phenomenon sampled is within the margin specified on the data sheet, but it is unknown with what probability that value is some distance k from the real value.

Lightweight temporal compression (LTC) is designed to compress data when sensor accuracy is expressed as a margin, and when the probability distribution of error is either uniform or unknown.

The key to LTC is to represent a sample by the value within the margin that maximizes compression. We simply *move* each sample up or down by some small amount (less than or equal to some maximum distance) to maximize compression.

The drawback to LTC is that it may convolute the original error distribution when that distribution is not uniform. However, this does not typically result in a loss of information because it is unusual for this original distribution to be known.

4.2. The Temporally Continuous Nature of Environmental Data

When one can characterize data with a model, the best compression technique is simply to return the parameters to that model that best fit the dataset. If no model is available, a general purpose technique must suffice.

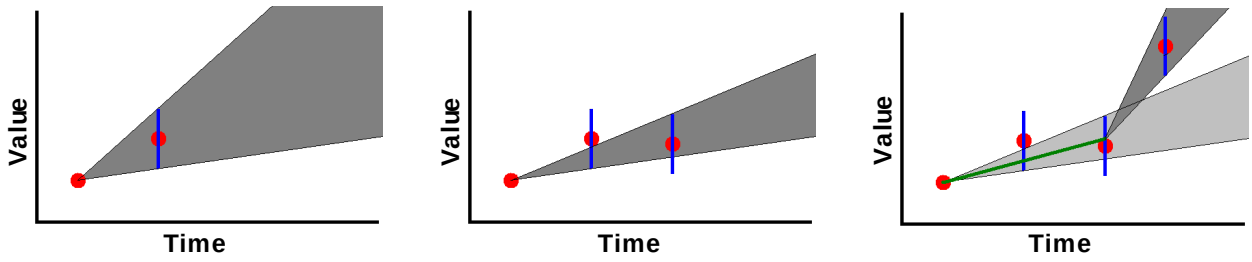
Environmental data such as temperature, and humidity have the nice property that they are usually continuous in the temporal dimension and at small enough time windows are approximately linear. Since models for spatially dense environmental data do not exist, we use this linear approximation to compress our data.

The linearity at a small timescale suggests that whenever a window of data has more than two data points, representing this data as a line uses fewer bits than a list of all discrete points. Furthermore, reconstructing the original data from such a line induces no additional error. Applying an arbitrary model-based compression scheme might result in a loss of important features in the dataset.

4.3. Environmental Data Modeling Assumptions

Environmental phenomena are inherently very complex and difficult to model accurately. Because we lack an accurate model for the various types of environmental data that we are interested in, applying an arbitrary model-based compression scheme might result in a loss of important features in the dataset. For example, linear least squares regression fits data to a line irrespective of whether the phenomenon is linear.

We don't assume a model for environmental data. Important environmental features come in a variety of



(a) (b) (c)

Figure 1. The algorithm explained. (a) LTC is initialized with two points. The first is fixed; the second is transformed into a vertical line segment and determines the set of all possible lines. (b) For each additional point, the set of all lines is reduced. (c) When a point does not fall within the set of all possible lines, the previous dataset is capped off and the procedure starts over.

forms including outliers, and changes in slope. Rather than guarantee that the reconstructed dataset matches the sample set in an aggregate statistical sense, we instead guarantee a match on a point-by-point basis, ensuring that any reconstructed data point is no more than some error margin from its corresponding sample point. This has the advantage of capturing outliers. Linear least squares, which uses the residual squared (R^2) as a quality metric, does not preserve outliers. Furthermore, LTC performs well even when the margin is set to be much less than the manufacturer's specified error margin.

4.4. The Compression Algorithm

LTC, described below, leverages temporal linearity to compress data. Our technique is analogous to run-length encoding, in the sense that we attempt to represent a long sequence of similar data with a single symbol. Where run length encoding searches for strings of a repeated symbol, we search for linear trends.

Let $r_i = (t_i, v_i)$ be a sample point with a tolerable margin of error e. A raw dataset $R = r_0, r_1, ..., r_j$ is translated into a stream of processed points, $S = s_0, s_1, ..., s_k$ where $k \leq j$ (typically $k \leq\leq j$). Let $L = l_0, l_1, ..., l_{k-1}$ be the set of line segments such that l_i is the line segment connecting the two points s_i and s_{i+1}. The piecewise continuous function defined by the segments of L approximates R such that no point in R is more than a vertical distance of e from the closest line segment in L.

To start this algorithm, we consider the first two points from the input data stream, r_1 and r_2. The first point (which we will call z) we leave unchanged. The second we transform into a vertical line segment (which we will call m) with x-coordinate of t_2, length $2e$, and centered at r_2. Thus, the y-coordinate of each point that makes up m, lies within the range $[v_2 - e, v_2 + e]$. Any line that passes through z and one of the points that make up m must come within a vertical distance of e from both r_1 and r_2. We can characterize the set of all possible lines (which we will call A) that go through z and one of the points of m by keeping track of z and the extremes of m (which we will call the upper limit UL and the lower limit LL).

The algorithm continues by considering the next point from the input data stream and transforming it into a vertical line segment (n) as described above. We calculate that part of A that also intersects n, and we update our UL and LL to reflect how our set has changed. This process continues until a vertical line segment (q) corresponding to a point in the data stream R does not intersect with any line in A. At his point A is *capped off*: all data from z up until the previous point is represented by z and the midpoint between UL and LL, which we will call MP. MP is then chosen as our new starting point z, and the extremes of q are chosen as the new UL and LL. The process continues in this fashion. This process is pictographically described in figure 1. The algorithm is as follows.

1. Initialization: Get the first data point, store into z. Get next data point (t_2, v_2), use it to initialize limits UL (UL is set to $(t_2, v_2 + e)$) and LL (LL is set to $(t_2, v_2 - e)$.
2. Calculate the *highLine* to be the line connecting z and UL.
3. Calculate the *lowLine* to be the line connecting z and LL.

4. Get next data point. Transform the point to a vertical segment using the margin e. Let ul be the highest point of the segment. Let ll be the lowest point of the segment.
5. If *highLine* is below the ll or if the *lowLine* is above the ul then goto 9, else continue onto the next step
6. If *highLine* goes above ul then set UL to be ul.
7. If *lowLine* goes below ll then set LL to be ll.
8. Goto 2.
9. Cap off: output z to the output data stream.
10. Set z to be the point midway between UL and LL.
11. Set UL to be ul.
12. Set LL to be ll.
13. Goto 2

Because the algorithm only needs to keep track of the starting point z, the upper limit UL, the lower limit LL, and the new data segment's limits (ul and ll), then its memory requirements are constant i.e. $\theta(1)$. In addition, the amount of work needed to process a single new point is bounded by some constant number of steps. Therefore, to examine n points takes $\theta(n)$ time. The storage and computational complexity of LTC is equivalent to linear least squares.

In the worst case the number of points that are outputted is no more than n and occurs when no line can be fit between more than two successive points. Lastly, because this algorithm is greedy it does not guarantee that the number of line segments used to represent a data set is minimal. However, an optimal algorithm would need to store the entire data set before a minimal set of lines could be calculated.

If the worst case does occur this might suggests that the error margin specified is either too small or the sampling rate of the data stream is too low. A saw-toothed data set resulting from sampling temperature every 12 hours, at noon and at midnight, is an example of the latter.

4.5. Compression vs. Longer Sampling Periods

When a user wants all of the data from a sensor network, compression and reducing the sampling rate are two ways of reducing communication over the radio. Dereasing the sampling rate, however, has the drawback of losing information between sampling points. With our compression it is possible to sample at high rates (i.e. have a good idea of what's going on at all times) while only sending a small amount of data.

If more than a few contiguous sampling points within a window can be accurately represented by a line, then the data has been oversampled. In theory, the sampling rate over that window could have been reduced to collect only two samples to achieve the same accuracy of our compression algorithm. The problem is that one has no way of predict how large the next time window (over which the data will appear to be linear) will be. The temperature, for example, could increase at a rate of one degree per minute for the next hour. Sampling temperature twice over the course of that hour would serve as an equivalently accurate summary to sampling once per minute. However, this clearly is not the case when the temperature is subject to seemingly random perturbations.

In terms of power consumption, sampling a sensor is relatively cheap when compared to the energy consumed during radio transmission. Therefore, LTC, in effect, makes it possible to sample at a high enough rate to detect any changes in environmental phenomena, but report at a lower instantaneous rate that is dynamically changing in response to the second derivative of our data stream.

Thus we get the benefit of a slow sampling rate (i.e. sending less data, less frequently) while maintaining the accuracy of a high sampling rate.

5. Experimental Results

Over the past year, the Continuous Monitoring System (CMS), an approximately 25-node, mote deployment has been collecting various types of environmental data including air temperature, humidity, and wind speed in a 3.5 hectare area of James Reserve [24]. We evaluate our lightweight temporal compression algorithm offline using streams of microclimate data collected using CMS. We use CMS data because it is abundant (each of the datasets we use has approximately 70,000 points), real, and collected in our target environmental context.

In evaluating LTC, we use three different sets of data: temperature, humidity, and wind speed. All three sets span the course of approximately one year with sampling periods on the order of minutes. These sets were chosen not only because these are the sensing modalities with which we intend to use LTC, but also because they are representative of three broad categories of environmental data sets. At the timescale of CMS, temperature is largely continuous, and very slowly changing. Humidity is not as well behaved as temperature and hence appears to be a wider-band phenomenon. Finally, wind speed can be very abrupt (i.e. discontinuous) as gusts of wind can pick up and

die down very quickly. Wind speed is similar in behavior to event detection sensors such as the rain bucket (used to measure the rate of rain fall).

5.1. Various Sensing Modalities

Figure 2 presents the percent decrease in bytes required to represent temperature, humidity, and wind datasets as a function of error. The y-axis shows the percent savings, calculated by $savings = (bytes_{raw} - bytes_{compressed})/bytes_{raw}$.

The x-axis represents error as a percent of the manufacturer's claimed margin of error. For example, the temperature sensor's data sheet specifies the margin of error to be $\pm 0.5^o$ C. Looking at the temperature curve, the point corresponding to 50% on the x-axis means that LTC achieves approximately 88% savings even when guaranteeing that all points on a dataset reconstructed from our compressed stream will be within $\pm 0.25^o$ C of the values of those points on the original raw stream. In the center of the x-axis is 100%. This is the point where the margin length used in LTC is set to margin specified by the manufacturer's data sheet. The other sensors margins of error as specified on their respective data sheets are: ± 0.25 mph for wind and $\pm 2\%$ for humidity.

In the limit, one would expect that as the tolerable error margin grows, the savings due to compression would approach 100%. (When a sensor has an infinite margin of error, any set of points can be represented by a single arbitrary line.) The interesting thing to note about our results is that all three curves level off at *very low* margins of error. In fact, most of the gain of LTC occurs well within the operating margin of error specified by the sensor manufacturers.

In all of our curves, the compression achieved when the tolerable margin of error was equal to the sensor manufacturer's margin of error (100% on the x-axis) is very significant. They are 3-to-1 for humidity, 12-to-1 for wind speed, and 20-to-1 for temperature. Even at *only* 10% of the manufacturer's margin, these figures are 1.5-to-1, 8-to-1, and 2-to-1 respectively.

The maximum margin of error between a raw data point and its corresponding post-compression reconstructed data point can be set arbitrarily for LTC. This serves as a tuning knob to adjust the tradeoff between compressed data size and accuracy. As accuracy decreases, high resolution components of a dataset may be lost. However, since temperature is a much lower frequency phenomenon than is the noise that temperature sensors produce, LTC might be used to cleanly filter noise, while maintaining a very low margin of error between the raw and reconstructed datasets. The

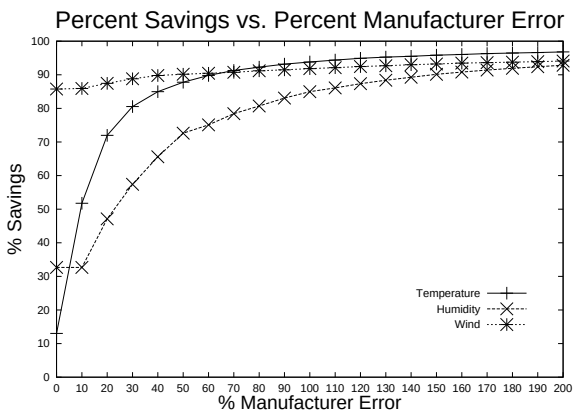


Figure 2. Percent savings resulting from compression of temperature, humidity, and wind data as a function of error. Error is specified as a percentage of the error margin listed on each sensor's data sheet.

pronounced "knee" at around 40% tolerable error in the temperature plot is evidence of this clean separation in the frequency domain.

Humidity is a wider-band phenomenon in frequency space than is temperature. Therefore, as we tune our error knob, we selectively filter a wider and wider band of high frequency information, which results in a more linear effect on savings. Unlike temperature compression, there is not a clean separation of signal and noise. Hence, while we achieve savings by filtering noise, the filtering of high frequency components of this signal might contribute to this savings as well. However, because of lack of separation, this signal loss is unavoidable in any compression scheme.

LTC behaves much like run length encoding when applied to wind data. Most of our wind dataset reads zero, because no wind is detected most of the time. Once in a while, wind is detected, but the speed of that wind is erratic. A long period of no wind can be represented as endpoints of a single line whose y coordinates are both 0. Erratic wind behavior, on the other hand cannot be compressed well. At points on the graph where the tolerable margin of error is high, we see 10% more savings than when the margin is low. This suggests that almost all of the savings is due to the fact that there are long periods of no wind and that such periods compress very well.

5.2. Various Compression Schemes

We compare LTC to lossless LZW and lossy wavelet compression. Results were obtained using two LZ77

Technique	Wind	Temperature	Humidity
LTC	91.8	93.8	72.6
LZW 12-bit	96.5	57.4	78.1
LZW 14-bit	96.8	64.3	81.4
Wavelet	89.7	91.1	83.5

Figure 3. Percent savings using LTC, LZW, and wavelet compression for the wind, temperature, and humidity datasets collected at James Reserve. The quantization threshold for the wavelet compression was set to produce the same root mean square error as with LTC. LTC performs about as well as its heavier-weight counterparts.

variants, one that uses a 12-bit symbol table and another that uses a 14-bit table. They are presented in figure 3.

The implementation of LZW used in our experiments [25] uses considerably more memory than the 4KB that a mote has available. The 12-bit version requires approximately 25KB and the 14-bit version requires almost 90KB. For this reason, without alteration, this implementation of LZW cannot run on motes. The wavelet compression was performed using Matlab's built-in functions.

Looking at the wind dataset, LZW achieves 96.8% compression (roughly 20:1). Although LTC never achieves this savings, it comes within 10% of it without introducing any error into our dataset (0% of the manufacturer error). If we induce some error, e.g., by setting the error tolerance to 200% (± 0.5 mph) of the manufacturer's error, LTC can get very close to the savings of LZW. LZW works so well on the wind dataset because it tries to match long substrings. Since the wind speed is most often zero miles per hour, there are long sequences of 0's throughout the data.

For the temperature dataset we achieve comparable savings when setting our tolerance to between 10% and 20%, well below the manufacturer's specified error. For humidity, we see largely the same as temperature. We achieve a savings of 33% without introducing any error, and achieve slightly greater savings than LZW when introducing error equivalent to the manufacturer's specified margin.

The manufacturers quoted error for wind, temperature, and humidity respectively are ± 0.25 mph, $\pm 0,5^o$ C, and $\pm 2\%$ relative humidity. The root mean square (rms) errors for the LTC values given in figure 3 respectively are 0.12, 0.29, and 1.20, all considerably below the manufacturers' quoted error.

LTC stands up pretty well to LZW and is computationally simpler. To further our understanding of LTC we compared it to a wavelet compression scheme. The form of wavelet compression used takes the entire dataset and transforms it into the frequency domain. Inputting the entire dataset at once maximizes the opportunity for compression, but clearly would not be feasible on mica motes. Run-length encoding is then used to compress the frequency components of the dataset. Frequencies that do not contribute much to the dataset are discarded. We adjusted the quantization threshold so that the rms error for wavelet compression was the same as for LTC compression when the margin was set to the manufacturer's specified error margin.

We then compared compression ratios. Looking at figure 3 we see that the savings achieved by LTC is comparable to that of wavelet compression. What is remarkable about this is that LTC performs comparably even though it uses a fraction of the computational and storage resources used by wavelet compression. The savings achieved by wavelet compression is dependant upon the size of the window over which it is performed. For our comparison, wavelet compression was given enough memory to hold the entire dataset. LTC only use $\theta(1)$ memory to perform its computation. In addition, wavelet compression depends upon performing the Fourier transform, which takes $\theta(nlgn)$ time to perform. Such complex mathematics may not be practical for the motes since they have an 8-bit processor and no hardware floating point unit.

6. Conclusion and Future Directions

The tunable lightweight temporal compression (LTC) scheme proposed in this paper is simple. To analyze n points takes $\theta(n)$ time and $\theta(1)$ space for intermediate calculations. It works on motes; it has already been implemented as part of the query processing engine in ESS. Despite its simplicity, it performs comparably to LZW and wavelet compression.

LTC has not yet been deployed at James Reserve; this is first order future work. Once it is part of a deployed system, we will further analyze its output and characterize its run-time performance. We also plan to evaluate its applicability to new datasets, and perhaps augmenting it in the process. In addition, we plan to explore variations of LTC to address data streams with a low sampling rate. Longer term, we plan to explore LTC's ability to filter noise for the purpose of performing feature extraction on the motes.

References

[1] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson., "Wireless sensor networks for habitat monitoring," in *ACM International Workshop on Wireless Sensor Networks and Applications*, Atlanta, GA, September 2002.

[2] Alberto Cerpa, Jeremy Elson, Deborah Estrin, Lewis Girod, Michael Hamilton, and Jerry Zhao, "Habitat monitoring: Application driver for wireless communications technology," in *Proceedings of the 2001 ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, April 2001.

[3] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks," in *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking*, Boston, MA, Aug. 2000, pp. 56–67, ACM Press.

[4] Samuel Madden, Michael Franklin, Joseph Hellerstein, and Wei Hong, "TAG: a tiny aggregation service for ad-hoc sensor networks," in *OSDI*, Boston, MA, dec 2002, vol. 1.

[5] Joseph Hellerstein, Wei Hong, Samuel Madden, and Kyle Stanek, "Beyond average: Towards sophisticated sensing with queries," in *IPSN '03*, Palo Alto, CA, 2003, vol. 1.

[6] Ph. Bonnet, J. Gehrke, and P. Seshadri, "Towards sensor database systems," in *Proceedings of the 2nd International Conference on Mobile Data Management*, Hong Kong, January 2001.

[7] Deepak Ganesan, Ben Greenstein, Denis Perelyubskiy, Deborah Estrin, and John Heidemann, "An evaluation of multi-resolution search and storage in resource-constrained sensor networks," in *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys).*, 2003.

[8] Deepak Ganesan, Deborah Estrin, and John Heidemann, "Dimensions: Why do we need a new data handling architecture for sensor networks?," in *First Workshop on Hot Topics in Networks (Hotnets-I)*, October 2002, vol. 1.

[9] Deepak Ganesan, Deborah Estrin, and John Heidemann, "Dimensions: Distributed multi-resolution storage and mining of networked sensors," *ACM Computer Communication Review*, vol. 33, no. 1, pp. 143–148, January 2003.

[10] Carlos Guestrin, Peter Bodi, Romain Thibau, Mark Paski, and Samuel Madde, "Distributed regression: an efficient framework for modeling sensor network data," in *Proceedings of the third international symposium on Information processing in sensor networks*. 2004, pp. 1–10, ACM Press.

[11] Milenko Drinic, Darko Kirovski, and Miodrag Potkonjak, "Model-based compression in wireless ad hoc networks," in *Proceedings of the first international conference on Embedded networked sensor systems*. 2003, pp. 231–242, ACM Press.

[12] S. Pradhan, H. Kusuma, and K. Ramchandran, "Distributed Compression in a Dense Sensor Network," in *IEEE Signal Processing Magazine*, March 2002.

[13] J. Kusuma, L. Doherty, and K. Ramchandran, "Distributed source coding for sensor networks," in *Proceedings of the IEEE Conference on Image Processing (ICIP*, October 2001.

[14] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," Tech. Rep., DEC SRC, 1994.

[15] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. on Information Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[16] 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 IEE International Conference on Computer Communications and Networks*, 2003.

[17] Lavanya Vasudevan, Antonio Ortega, and Urbashi Mitra, "Application-specific compression for time delay estimation in sensor networks," in *Proceedings of the first international conference on Embedded networked sensor systems*. 2003, pp. 243–254, ACM Press.

[18] Christiano Pereira, Sumit Gupta, Koushik Niyogi, Iosif Lazaridid, Sharad Mehrotra, and Rajesh Gupta, "Energy efficient communication for reliability and quality aware sensor networks," Tech. Rep. TR03-15, University of California at Irvine, Apr. 2003.

[19] Michael Hamilton, "James San Jacinto Mountains Reserve," `http://www.jamesreserve.edu`.

[20] "Stargate," `http://staging.xbow.com/Products/productsdetails.aspx?sid=85`.

[21] John Heidemann, Fabio Silva, and Deborah Estrin, "Matching data dissemination algorithms to application requirements," in *Proceedings of the first international conference on Embedded networked sensor systems*. 2003, pp. 218–229, ACM Press.

[22] G.J. Pottie and W.J. Kaiser, "Wireless integrated network sensors," *Communications of the ACM*, vol. 43, no. 5, pp. 51–58, May 2000.

[23] A.A. Abidi, G.J. Pottie, and W.J. Kaiser, "Power-conscious design of wireless circuits and systems," *Proceedings of the IEEE*, vol. 88, no. 10, pp. 1528–45, October 2000.

[24] Michael Allen, Thanos Boulis, Kevin Browne, Naim Busek, Vlad Bychkovskiy, Deborah Estrin, Michael Hamilton, Sheri Lubin, Mohan Mysore, Eric Osterweil, Mohammad Rahimi, John Rotenberry, Stefano Soatto, Mani Srivastava, Michael Taggart, and Michael Wimbrow Thomas Unfried, "Habitat Sensing at the James San Jacinto Mountains Reserve," `http://www.cens.ucla.edu/Education/Posters/NSF%20Poster%20Session-June%%204/PosterFiles/24_James_Reserve.pdf`.

[25] Mark Nelson, "LZW Data Compression," `http://www.dogma.net/markn/articles/lzw/lzw.htm`.