Data-Aware, Resource-Aware, Lossless Compression for Sensor Networks

Rachel Cardell-Oliver[1], Stefan Böttcher[2], and Christof Hübner[3]

[1] The University of Western Australia, Perth, Australia
rachel.cardell-oliver@uwa.edu.au
[2] Universität Paderborn, Germany
stb@uni-paderborn.de
[3] University of Applied Sciences Mannheim, Germany
c.huebner@hs-mannheim.de

Abstract. Compressing sensor data benefits sensor network applications because compression saves both transmission energy and storage space. This paper presents a novel lossless compression algorithm for sensor networks that is both data-aware and resource-aware. The DARA algorithm provides high compression ratios and also has a small memory footprint and efficient execution well within the range of sensor nodes. It is demonstrated that data-awareness, that is exploiting the structure of sensor data, is an important contributor to compression performance. The practicality of the DARA algorithm is demonstrated by an application in which sensor nodes use a phone modem to transmit a daily digest of nine sensor data streams in a single SMS message.

1 Introduction

Compressing sensor data benefits sensor network applications because compression saves transmission energy and storage space. Energy is saved when the quantity of data transmitted by network nodes is reduced. Compression also reduces on and off-node storage requirements so contributing to the goal of building long-life, unattended sensor networks. Any sensor network application that can tolerate delays in the delivery of their data can benefit from on-node compression. Such applications include environmental networks, particularly in remote areas; networks using opportunistic data collection by mobile data mules; bio-data sensed by body implant sensors; networks that transmit code for node reconfiguration; and applications that require efficient off-network storage of their data. Even for delay-intolerant applications, compressing current data can save transmission energy and storage since shorter messages are generated.

Compression algorithms for text and images have been widely studied, but only a few have been proposed that are suitable for the constraints of sensor networks. We argue that successful compression of sensor data streams requires both *data-awareness* of the structural properties of the raw data and *resource-awareness* of the constraints of the sensor nodes on which compression algorithms will be run.

This paper presents the design and evaluation of a generic algorithm, DARA, for on-node, lossless compression of sensor readings. The main contributions are as follows. (1) DARA is data-aware: it maps raw sensor data streams (that is, timed series

P. Demeester, I. Moerman, and A. Terzis (Eds.): EWSN 2013, LNCS 7772, pp. 83–98, 2013.

of readings from a particular sensor) to an equivalent form that maximises compression. (2) DARA is resource-aware: it uses Huffman coding with small codebooks, and allows for previously unseen or erroneous readings, as well as lost transmissions. (3) DARA is flexible and practical: analysis of different types of sensor data streams and evaluation of an implementation show that it offers better compression ratios than existing algorithms while still having low space and time complexity, and being adaptive to real-world conditions.

The paper is organised as follows. Section 2 reviews related work on compressing sensor data streams and gives an overview of our approach. Section 3 investigates data-awareness by testing the compressibility of different data models. Section 4 discusses the resource-aware features of the DARA algorithm. To illustrate the practicality of our algorithm, Section 5 presents the implementation of a sensor network application that reads heterogeneous sensor data streams and delivers compressed data once per day in an SMS message. This application requires a high degree of compression, and so is a good demonstration for the performance of the DARA algorithm.

2 Background and Related Work

Compression algorithms transform a stream of *input symbols* into a stream of *encoded symbols* by mapping one or more symbols in the input onto a code word [2]. The compression ratio *CR* for a particular message m is defined by

$$Compression\ Ratio : CR(m) = \frac{compressed\ size(m)}{original\ size(m)}$$

i.e., uncompressed data has a CR of 1.0. CR depends on both the lengths of the code symbols and the length (in symbols) of the input data. Our goal is to consume very little energy to achieve a low compression ratio for data from typical environmental sensor network applications.

Compression algorithms are either lossless or lossy. The former reduce the size of data to be transmitted whilst also maintaining accuracy, while the latter compress data within some error bound of the actual observed values. This paper focusses on lossless compression since accuracy is either required or at least can be considered as an upper bound for lossy algorithms. There have only been a few studies of lossless compression algorithms that are suitable for efficient execution on sensor network hardware and tailored to sensor network data.

Marcelloni and Veccio's simple lossless entropy compression algorithm (LEC) uses a generic Huffman code with 14 code words to encode the high order bits of each input symbol [7]. Then, each prefix code is concatenated with uncoded low order bits for the symbol. This approach has the advantage of a small codebook that is able to represent any 16-bit input symbol. LEC offers better compression ratios than dictionary-based algorithms such as S-LWZ [10] as well as having lower memory and energy use [7]. However, the generic code used in LEC does not perform as well as a pure Huffman code for difference data. This paper introduces an approach that gives better compression than LEC but has a similarly small codebook. An extensive evaluation of

real sensor data is used for examining the effects of data representation and codebook size independently, thus showing the contribution of each to successful compression.

Reinhardt et. al. [8] proposed an adaptive coding scheme for creating and adapting a small Huffman code that is updated as new symbols are seen. They analysed its performance on four data sets. The algorithm is "application-agnostic" in that the method is not tailored to any particular application. However, generic algorithms still have parameters that must be set, and we argue in this paper that data-awareness of the application leads to better compression ratios without any significant additional programming efforts for the designer.

S-LWZ is a dictionary-based, lossless compression algorithm for sensor networks based on Lempel-Ziv coding [10]. Sadler and Martonosi investigated the effect of block size, dictionary size, a mini-cache, and a dictionary overflow strategy on the performance of S-LWZ. However, both the compression ratio and performance costs of S-LWZ are higher than statistical coding algorithms such as LEC [7].

Some lossy compression algorithms can achieve lossless compression when their error bound is set to zero. For example, run length encoding is used by K-RLE, where runs of the same symbol, within an error bound K, are replaced by a (symbol,run-length)-pair [5]. In lightweight compression, sequences of microclimate sensor readings are replaced by a line segment approximation [12]. Latent variables can be used for estimating blocks of missing data [13]. However, although these algorithms have high compression ratios for non-zero error thresholds, they do not perform as well as statistical codes when lossless compression is required [7].

Algorithms designed for sensor network nodes must offer not only a high compression ratio, but also be able to run within the resource constraints of limited memory and processing power of sensor nodes. This constraint rules out a number of general-purpose compression algorithms, such as bzip [3], because of their relatively high memory and processing costs [10,1,7]. In fact, Barr and Asanović [1] have shown that using some of these algorithms can result in a net energy *increase* when compression is applied before transmission. And Reinhardt et. al. [9] showed that full adaptive Huffman coding may exceed the memory of sensor platforms, but performs no better than RLE which has minimal memory requirements. Another aspect of compression in sensor networks is fault tolerance to message loss. Guitton et. al. show that standard compression techniques may perform worse than no compression when a link has a greater than 10% packet loss rate [6].

Existing resource-aware compression algorithms designed for sensor networks focus on generic solutions, thereby excluding techniques that tailor the algorithm to the data to be compressed. In this paper, we propose and evaluate an algorithm that is both data-aware and resource-aware. Using experimental sensor data sets, we explain methods for determining the parameters of this algorithm based on analysis of the observations to be compressed.

The DARA algorithm is trained on a representative set of sensor readings together with user-provided constraints, such as target compression ratio, transmission block size, and memory limits. In the offline training stage, DARA produces, 1) a data transformation function and 2) codebooks for compression, as follows:

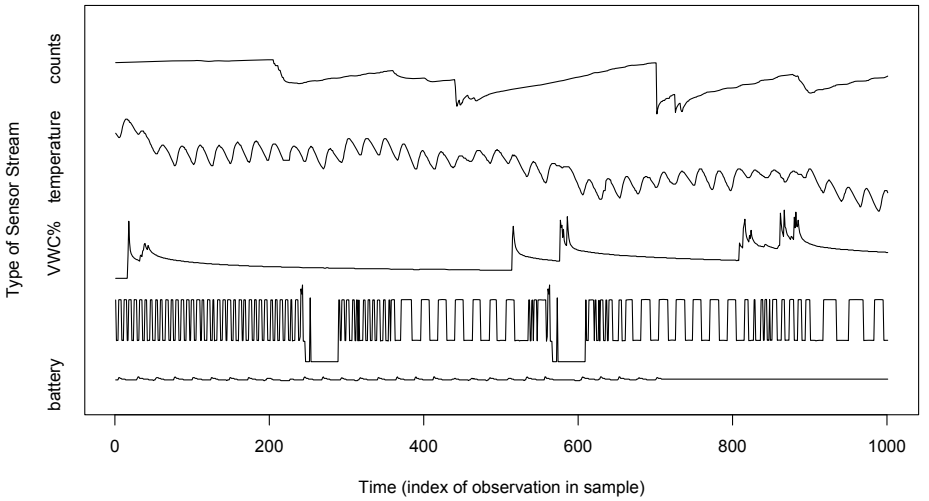

Fig. 1. Sample sensor data stream values from the benchmarks, 1000 observations each with y-axis scaled for readability

Stage 1: Data-Aware Transformation	
Given:	Training data as a matrix of raw sensor observations.
Generate:	A pipelined function to convert raw data into an equivalent but more compressible form.
Goal:	Minimize entropy and the number of symbols in the transformed data.

Stage 2: Resource-Aware Compression	
Given:	Transformed data from stage 1.
Generate:	Codebooks and block size for data compression.
Goal:	Minimize the size of transmitted data and codebook memory.

The DARA algorithm can be generalised for use with a number of different network models. For the evaluation in this paper, a single node was used with a phone modem sending daily digests by SMS to a nominated phone account. This node was directly connected to nine different sensors. DARA can also be used to determine compression parameters in networks where cluster head nodes receive sensor readings over the radio from their neighbours, and then transmit a compressed digest of all the readings to a base station.

3 Data-Aware Compression

In this section, we consider properties of sensor network data that can be exploited for compression and present an algorithm for pre-processing sensor data streams to optimise compressibility. The goal for a data-aware algorithm is to choose a representation for sensor data that maximises its compressibility without loss of accuracy.

Table 1. Sensor data benchmarks and their features

Data set ID	counts	hetero	calgeo
Number of Observations	14,116	24,408	5,000
Period of Observations	60 minutes	60 minutes	NA
Bits per observation	16	16	32
Entropy (bits per symbol)	11.92	7.32	12.05
Compression ratio CR	0.74	0.46	0.38
Number of symbols NS	5,973	300	4,618
Number of sensor data streams	4	9	1
Types of sensor data streams	frequency (4)	soil moisture (4) temperature (4) battery voltage (1)	bytes (4)
Range of sensor values	3.4e4 to 5.2e4	1.4 to 16.6 11.3 to 26.8 5.4 to 6.0	0 to 4.1e9

As a basis for analysis, we use a general model for sensor readings. Sensor data is represented by a matrix $M = [m_{i,j}]_{r \times c}$ containing symbols for observed sensor readings. Each of the r rows of the matrix contains readings from different sensors taken at the same time. A timestamp is associated with each row. Each of the c columns of the matrix contains consecutive readings from a particular sensor. The matrix can be heterogeneous: that is, different columns represent sensors that may sense different phenomena, such as soil moisture, soil temperature, humidity, sap flow, node battery, and so on.

3.1 Sensor Data Benchmarks

To explain and evaluate our algorithm, we have selected real-world, public domain benchmarks from long-running environmental WSN deployments [14] and a seismic data set [4]. The sensor data streams in these benchmarks include: soil moisture as frequency counts, soil moisture converted to scientific units (volumetric water content), soil temperature in degrees C, battery voltage, and 32-bit seismic observations. Figure 1 shows (scaled) samples of 1000 observations from each type of sensor in our benchmarks. This figure illustrates the wide variety of patterns seen in observed phenomena. Table 1 summarises the characteristics of each benchmark: its entropy (in average bits per symbol), from which we calculate the compression ratio that would be achieved using a perfect code. The *heterogeneous* data set contains soil moisture, temperature, and battery readings. The other sets are *homogeneous*, containing observations from only one type of sensor.

The following subsections introduce functions for transforming sensor data without loss of accuracy. These functions are pipelined together to transform raw data streams for compression. The choice of data representation is important because it can have a significant effect on the compressibility of the data.

Table 2. The counts benchmark as raw frequencies and converted to Volumetric Water Content (VWC%) with 0, 1, or 2 significant figures

Units	Significant Figures	Compression Ratio	Unique Symbols
frequency	0	0.74	5,973
VWC%	2	0.62	1,774
VWC%	**1**	**0.44**	**235**
VWC%	0	0.25	26

3.2 Engineering Units

A sensor reading, either digital or converted from an analog input, is typically represented by one or more bytes. For example, a positive integer represents a raw reading such as a dielectric soil moisture frequency, 16 bits are used for representing the sign and value of a temperature reading with one decimal place accuracy. Raw sensor readings are converted into engineering units using (for example) a polynomial conversion function. The result of the conversion is rounded to significant figures (SF) so that the measurements are reported to the accuracy of the sensing instrument that made the measurement. For example, in Table 1 each sensor data stream is represented accurately using one decimal place, i.e., 1 SF is appropriate for the accuracy of the sensors (shown in bold font), 0 SF gives lossy compression, and 2 SF is unjustified accuracy.

3.3 Temporal Correlation

Consecutive sensor readings from the same sensor are strongly correlated for most environmental phenomena. That is, the difference between a sensor reading and its temporal successor is a small value. A block of r rows of sensor readings can be represented by a timestamp followed by 1 row of values and $r - 1$ rows of differences $d_{i,j} = m_{i+1,j} - m_{i,j}$ [7]. For example, a sample block of given soil moisture readings preceded by a timestamp from the counts benchmark is shown in the left hand box of Figure 2. The right hand box shows the same block represented by a time stamp in row 1 and an *anchor* of values (in VWC% engineering units) in row 2, followed by temporal *differences* for the remaining 3 rows of the block.

For most sensor data streams, using differences significantly improves compression ratios. However, the difference representation does not improve compressibility for the calgeo data since the periodic alternating pattern of that stream has a similar numbers and frequencies for the symbols in both its raw and temporal difference forms.

In order for a receiver to be able to recreate engineering values from an encoded difference matrix an *anchor row* of values $m_{1,1}, \ldots m_{1,c}$ together with a time stamp is transmitted with each block of difference data as shown in the example of Figure 2. Efficient methods for encoding both anchors and differences are discussed in Section 4.

3.4 Sensor Type and Sensor Location Correlations

The main distinction between the columns of a block of heterogeneous sensor readings is the different phenomena being measured by the sensor data streams. For example, the

2011-04-25 22:04			
24.5	32.5	36.4	33.4
24.4	32.5	36.4	33.3
24.4	32.5	36.4	33.3
24.4	32.5	36.4	33.3

2011-04-25 22:04			
24.5	32.5	36.4	33.4
-0.1	0.0	0.0	-0.1
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

Fig. 2. Absolute vs. difference representation of sensor data

hetero data set contains 3 types of data: soil moisture readings (in percentage volumetric water content); soil temperature (in degrees C); and battery voltage readings (in volts).

To take advantage of differences between different data types, we allow the data set D to be partitioned into subsets D_1 to D_k. The compression ratio for the whole data set is the *weighted sum* of the CRs for each subset. The total number of symbols that must now be coded is the *total sum* of symbols in each D_i. This step offers a trade-off between reducing the overall compression ratio and usually increasing the total number of symbols. The identity function is always one of the candidate splitting functions.

3.5 Data-Aware Compression Summary

Stage 1 of DARA generates a data transformation function that maps raw sensor data streams to an equivalent, but more compressible, form. The full process is summarised in Figure 3. Table 3 shows how the algorithm works out in practice for the benchmark data sets. Recommended representations are highlighted in bold in Table 3. Comparing rows Differences and Sensor Type, we see that for the hetero data set splitting on the type of sensor decreases the compression ratio from 0.15 to 0.11, *but* the number of symbols to be coded increases from 78 to 119. In short, the gain in compressibility may not justify the resource costs of learning and maintaining separate codebooks for a larger symbol set. For calgeo, the compression ratio actually increases from 0.38 to 0.51, but the number of symbols falls from 4530 to 46 when 32-bit code words are broken down into 4 x 8-bit codewords. Although the number of bits per symbol falls for each byte stream, the size of the data matrix is larger. In DARA, a weighted utility function is used for determining the best trade-off between compression ratio and number of symbols for transformed data given user targets for these values. Typically, the weight $\alpha = 0.5$, but this can be changed, depending on the application, to favour higher compression or reducing the symbol set.

4 Resource-Aware Compression

Stage 2 of DARA determines the parameters for compressing the transformed sensor data from Stage 1. The design goals for a resource-aware algorithm are to minimise the algorithm's memory use and execution time when the algorithm is run on sensor node hardware. The DARA compression algorithm is based on Huffman coding with modifications to address the resource constraints of sensor nodes.

Input: Training data $R = [m_{i,j}]_{r \times c}$ for the sensor network application
Output: Recommended transformation function $f = f_S \circ f_D \circ f_E$

Process:
$f_E(R) = E = [e_{i,j}]_{r \times c}$ with $e_{i,j}$ in engineering units ;
$f_D(E) = D = [d_{i,j}]_{r-1 \times c}$ with $d_{i,j} = e_{i+1,j} - e_{i,j}$;
do
TEC = ask user for target entropy compression ratio ;
TNS = ask user for target number of symbols ;
α = ask user for a weighting of *TEC* and *TNS* ;
TU = ask user for target utility value ;
F_S = ask user for a set of candidate split functions ;
u = *maxValue* ;
for each $f_{Si} \in F_S$ {
$\langle D_1, \ldots, D_k \rangle = f_{Si}(D)$; // get the split sub-matrices
for each $j \in [1, ..., k]$ { $w_j = |D_j|/|D|$; } // calculate weight of each sub-matrix
$cr = \Sigma_{j=1}^{k}$ *CR*$(D_j) \times w_j$; // weighted compression ratio
$ns = \Sigma_{j=1}^{k}$ *NS* (D_j) ; // total number of symbols
$u_i = (\alpha \times cr/TEC + (1 - \alpha) \times ns/TNS)$; // the utility value
if ($u_i \leq u$) { f_S=f_{Si} ; u=u_i ; } // select f_{Si} to minimize u
}
while ($u > TU$) ;
return $f = f_S \circ f_D \circ f_E$;

Fig. 3. Algorithm to generate a Data-Aware transformation function

4.1 Huffman Codes for Sensor Data

In order to encode an input stream efficiently, a compression algorithm requires a model for the symbols in that stream. Both context free models (e.g. in English text e is the most common letter and t is the next most common) and context sensitive models (e.g. q is usually followed by u) have been proposed, particularly for compressing text [2]. Huffman code is a context free algorithm. It maps input symbols to code words of different lengths based on the frequency of the input symbol, and it can achieve compression approaching the Shannon entropy limit.

A canonical Huffman code for the hetero benchmark requires a codebook with 300 symbols and codes of length 1 to 15 bits. Both differences and anchors are encoded. The advantage of a full Huffman code is that the compression ratio achieved can be equal to the best possible. However, this solution is not well suited for resource constrained sensor network nodes for a number of reasons:

- The codebook for 300 symbols is too large. Codebooks should be as small as possible on resource constrained nodes.
- The data has been over-fitted to the training data. Field data may contain different values that occur with different probabilities.
- New symbols that were not in the training set can not be encoded.

- Encoding will be sub-optimal if the frequencies in operational data differ from the training data.
- No allowance is made for missing data and erroneous sensor readings.

Stage 2 of DARA, described in the following sections, is an extension of the basic Huffman algorithm that is designed to address these problems.

Table 3. Data-Aware Analysis of the compression ratio (CR) and number of symbols (NS) for three benchmarks. The recommended representation for each benchmark is highlighted in bold for given targets and utility function.

Benchmark	counts			hetero			calgeo		
Utility $\alpha = 0.6$	TCR=0.25, TNS = 100			TCR=0.25, TNS = 100			TCR=0.60, TNS = 100		
Representation	CR	NS	Utility	CR	NS	Utility	CR	NS	Utility
Raw	0.74	5973	25.67	0.46	300	2.30	0.38	4618	18.85
Engineering	0.43	235	1.97	0.46	300	2.30	0.38	4618	18.85
Differences	**0.06**	**38**	**0.29**	**0.15**	**78**	**0.67**	0.38	4530	18.50
Sensor Type	0.06	79	0.46	0.11	119	0.74	**0.51**	**46**	**0.69**

4.2 Reducing the Codebook Size

The codebook for the hetero benchmark in engineering units has 300 entries: one for each different symbol in the training data. The codebook for hetero differences has 78 symbols. Even this may be too high an overhead for a memory resource-limited sensor node. A solution to this problem is to truncate the codebooks: use a Huffman code for only the most common input symbols and transmit the other symbols uncoded. The set of common symbols that are assigned Huffman codes is given by symbol set S, and a code symbol $\mathcal{U}$ is chosen to distinguish the uncoded symbols. For any symbol $s \notin S$, s is coded by $\mathcal{U}.s_b$. That is, the concatenation of the prefix $\mathcal{U}$ and symbol s restricted to b bits. A simple choice for $\mathcal{U}$ is to use the next Huffman code in the canonical sequence after assigning codes for the symbols in S. The number of bits b should be just large enough to encode all legal symbols: for example, 9 bits are sufficient for hetero differences.

Figure 4 (left) shows the relationship between codebook size and the compression ratio achieved using truncated codebooks. For the hetero benchmark, there is only a small penalty to pay in compression ratio by reducing the codebook size. For hetero differences, using a codebook with 20 symbols gives CR of 0.16, compared with CR of 0.15 for the full 78-symbol codebook. An alternative labelling strategy is to append a single bit to every codeword to distinguish between coded and uncoded values [8]. However, when symbols in S account for the majority of inputs, then our strategy requires fewer bits overall since the unknown symbol prefix will only rarely be used.

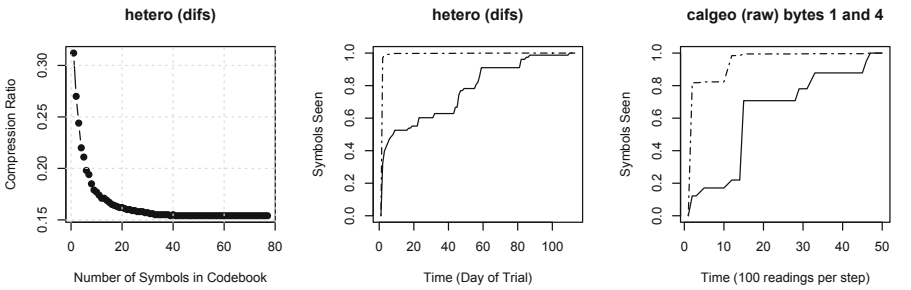

Fig. 4. Codebook size vs compression ratio for hetero differences (left), time versus proportion of symbols seen (solid line) and cumulative frequency of symbols (dotted line) for hetero differences (middle), and calgeo bytes (right) benchmarks.

4.3 Encoding New Observations

Another question is how to encode new symbols that are observed during operation of the sensor network, but that did not appear in the training data. This is known as the zero-frequency problem [2]. In DARA, new symbols are handled in the same way as rare symbols: each new symbol is encoded using the not-yet-coded prefix code. Figure 4 (middle and right) shows the proportion of different symbols seen versus time for the hetero and calgeo benchmarks. The results show a steady growth of new symbols seen over time. The dotted lines show the cumulative frequency distribution for number of symbols over time. After the first day (hetero) of 112 days, 97% of all sensor readings have already been seen, and 81% after the first 100 readings (calgeo) of 5000. Thus, although new symbols are seen later, such symbols occur with very low frequency and so can safely be treated in the same way as other rare symbols.

4.4 Missing and Erroneous Data

Sensor data is notoriously noisy: observed values may be out of range, out of context, or missing as a result of sensor or communication failure. We assume that erroneous observations can be identified by the sensor node and are reported using a unique symbol $\mathcal{E}$. When interpreting raw data blocks, $\mathcal{E}$ is replaced with an estimated value: the most recent in-range observation from the same sensor data stream. When calculating differences $\mathcal{E}$s have the same value as their predecessors, and so their difference value is 0. A special symbol $\mathcal{E}_0$ is used for estimated differences. If missing data occurs with high frequency then $\mathcal{E}_0$ may have its own symbol in the Huffman codebook. Otherwise $\mathcal{E}_0$ and $\mathcal{E}$ will be encoded using a prefix code in the same way as all other not-yet-coded symbols. The receiver can choose either to filter out estimated observations or to treat them as estimates for real observations.

4.5 Adaptive Coding for Changing Frequencies

When compressing a stream of sensor network data over time, the frequency of input symbols may change from their frequency in the training data. Frequency changes either occur because there is a change in the distribution of sampled data (for example, because of the time of the year in an environmental network) or because the original training data was not representative of the application data. Such changes lead to sub-optimal compression. Adaptive Huffman coding solves this problem.

In DARA, a simplified adaptive Huffman algorithm is used [11]. This algorithm does not guarantee an optimal tree. However, because of its simplicity it is better suited to implementation on resource constrained sensor nodes than the full adaptive Huffman algorithm used in other studies [8]. The sender and receiver follow the same process as follows. Start with a codebook created for the observed data. Both, transmitter and receiver store 3 columns: input symbol, a count of times the symbol has been seen, and the Huffman code for that symbol. The rows are ordered by the count column. If frequencies change, so that the ratio between symbol probabilities in rows i and $i+k$ is lower than the swap ratio threshold $SR \in [0,1]$, then swap the name and count of those rows, leaving the codewords as they were.

How does the receiver keep track of changes in the sender's codebook? If we have a feedback mechanism from the receiver to the sender, the receiver can require the sender to send a lost message again, and we have no problems at all (except for message delays). If we have no feedback mechanism from the receiver to the sender, the sender will not know which messages are lost. Now many strategies to treat this situation are possible, of which we prefer the following:

1. The sender can repeatedly send codebook changes after at most every N messages or, without extra costs, as part of each coded messages that, together with the codebook changes, does not exceed the allowed coded length.
2. After a message loss, the receiver can treat further received messages as being lost or depending on the thresholds, can treat some of these messages' values as being guaranteed and other values as being estimates only, until the receiver is again sure about the codebook.
3. To further reduce situations where the receiver is unsure about the codebook, the sender could enumerate its codebook versions and send a few bits with each block denoting the actual codebook version, such that the receiver can see whether or not the version is the same as his last codebook version even if a sequence of messages was lost.

In this sense, transmitted code blocks retain their property of being idempotent. That is, each message can be correctly decoded without reference to the preceding messages. The receiver can correctly adapt its codebook as soon as it receives a message from the sender even if a sequence of messages has been lost.

4.6 How to Encode Anchor Data

Huffman codes are inefficient when the probabilities of most input symbols are similar (rather than negative powers of two) and when there are a large number of symbols. For

example, when differences are used for representing data, then a known *anchor message* is needed in order to recover the original observed values. An example anchor message for the hetero benchmark is: `<12-06-25 16:00 3.5 8.2 8.7 6.8 15.6 15.0 15.2 14.8 5.6>` comprising a time stamp of 5 values followed by 9 sensor values. Anchor messages have a large number of symbols from different sensor data streams and the frequency distribution of those symbols is relatively uniform. Rather than using a Huffman code for anchors, we propose the following code:

1. Inputs are messages $< s_1, \ldots, s_n >$ where each symbol s_i may come from a different distribution.
2. For each field i, experts determine $scale_i$ to map symbol values to an integer range, min_i, max_i, and we calculate $range_i = max_i - min_i + 1$ for the scaled values. For example, possible battery values in the hetero benchmark are 3.5 to 6.5 Volts, giving $scale_i = 10$, $min_i = 35$, $max_i = 65$, and $range_i = 31$ for this application.
3. The number of bits required to represent field i is $b_i = \lceil log_2(range_i) \rceil$. For example, $b_i = 5$ bits are required to represent battery readings within a range of 31.
4. The code for symbol s_i is $int2bin((s_i \times scale_i) - min_i, b_i)$. For example, battery value 5.7 is represented by (57 - 35) = 22, in 5-bit binary as 10110. Of course, the scaling and the range computation have to consider the accuracy of the sensed values, i.e., if the battery accuracy is 0.05 Volts with allowed values from 3.5 to 6.5 Volts, we have to represents 61 possible values, needing a 6-bit code.
5. The anchor codebook consists of three n-vectors: $scale$, min and b.

Anchors encoded in this way for the hetero benchmark require 104 bits per anchor which is 7.4 bits per symbol (b/s). This is slightly higher than the entropy of 6.9 b/s for this data, but has the advantage of a very small codebook of 3 by 14 entries compared to a Huffman code for over 300 unique anchor symbols. Anchor coding is, however, still much less efficient than Huffman coding of differences for the remainder of the data where the CR is 0.16.

4.7 Resource-Aware Compression Summary

Figure 5 summarises the resource-aware steps of the DARA algorithm. The overall compression ratio achieved is a combination of the ratios for the anchor and Huffman codes. This target compression ratio will be slightly higher than the entropy target used in Stage 1 of the algorithm. A block size for transmitted messages is chosen to achieve the target compression ratio and satisfy codebook memory constraints.

5 Evaluation: An SMS Sensor Network

In order to evaluate the practicality of DARA, we implemented a sensor network that reports its data using SMS messages. A mobile phone sensor node was built and deployed using: a high-performance, low power microcontroller (ATmega 1284p), GSM GPRS quad-band module (TELIT GC 864-QUAD); 4 sensor ports; on-board temperature and battery sensor; real time clock; SD card; and serial flash. The node was powered by rechargeable batteries and a solar panel. The whole application, including encoding,

Input: Training data $R = [m_{i,j}]_{r \times c}$ for the sensor network application
Output: Recommended codebooks $C_1, \ldots, C_k$, A and block size B

Process:
$f = f_S \circ f_D \circ f_E$; // calculated for R as specified in Figure 3
$\langle D_1, \ldots, D_k \rangle = f(R)$; // transform data for Huffman encoding
for each $i \in [1, ..., k]$ {
  $w_i = |D_i|/|D|$; //calculate weight of each sub-matrix
  $H_i =$ *HuffmanCodebook*(D_i) ; // calculate a full Huffman codebook for each D_i
}
$f_E(R) = E = [e_{i,j}]_{r \times c}$ with $e_{i,j}$ in engineering units for anchor rows ;
$A =$ *AnchorCodebook*(E) ; // calculate anchor codebook
crA = *CR*$(A(E))$; // calculate compression ratio for anchor encoding
TU = ask user for target utility value ;
u = *maxValue* ;
do {
  TCR = ask user for target compression ratio ;
  TCS = ask user for target total codebook size ;
  B = ask user for target transmission block size ;
  α = ask user for a weighting of *TCR* and *TCS* ;
  $\langle n_1, \ldots, n_k \rangle$ = ask user for truncated codebook sizes with $n_j \leq |C_j|$;
  for each $j \in [1, ..., k]$ {
    $C_j =$*truncate*(H_j, n_j) ; // truncate each full codebook to n_j entries
  }
  crD= $\Sigma_{j=1}^{k}$ *CR*$(C_j(D_j)) \times w_j$; // weighted compression ratio
  cr = $(1 \times$ *crA* + $(B-1) \times$ *crD*$)/B$; // overall compression ratio
  $ns = |A| + \Sigma_{j=1}^{k} |C_j|$; // total size of all codebooks
  $u_c = (\alpha \times cr/TCR + (1-\alpha) \times ns/TCS)$; // the current utility value
  if ($u_c \leq u$) { $u = u_c$; } // select codebooks to minimize u
}
while ($u >$ *TU*) ;
return $C_1, \ldots, C_k,\ A,\ B$;

Fig. 5. Algorithm to generate Resource-Aware compression codebooks

uses 19.0% of the program memory and 43.8% of the data memory, easily meeting the requirement of resource-awareness for sensor node hardware since this SMS application has no requirements for routing or other network management code. Our evaluation used 9 sensor data streams: 4 soil moisture, 4 soil temperature, and 1 battery voltage sensor. Readings were taken once per hour, and reported in an SMS message once every 24 hours. Additional messages to report alarm conditions outside this schedule could also be sent as required, but are not part of this evaluation.

SMS messages are written using a subset of the 128 ASCII symbols. However, there are some differences in the SMS codes used by different manufacturers since selected ASCII symbols are actually transmitted as 2 characters. So we restrict our code to 64 non-controversial character symbols each of which can be represented by a 6-bit binary string. The binary Huffman encoding of a coded block produced by DARA is mapped to an SMS transmittable stream by replacing 6 bits at a time with one of the selected

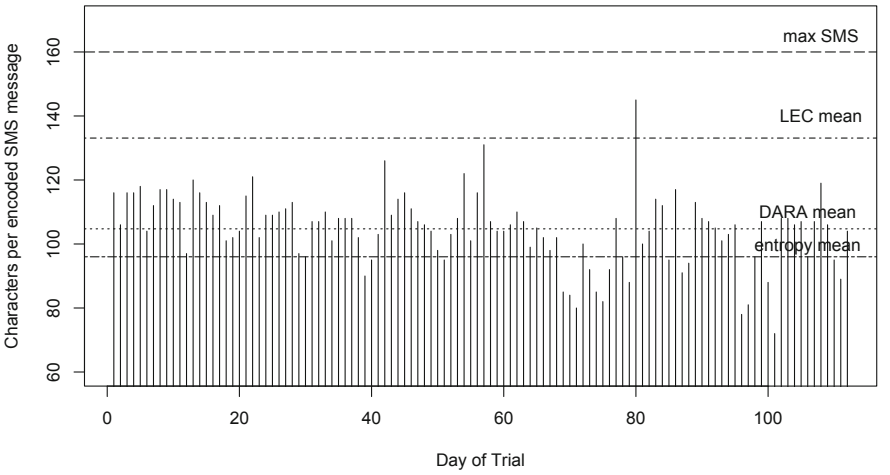

Fig. 6. Code length in characters of each encoded message for evaluation data set. Horizontal lines show the mean message lengths for DARA and related encodings.

ASCII SMS symbols. In this way, a 160 character SMS message can encode a block of up to 960 coded bits.

Using the DARA algorithm, a single codebook for differences was defined with 20 entries, block size $B = 24$ hours, and an anchor codebook. Figure 6 shows the length of the resulting SMS encoded message (in characters) for each day of the evaluation. The mean message length was 104.7 characters which is a compression ratio of $CR = 0.24$. The shortest encoded message had 72 characters and the longest 145, i.e., no messages in the evaluation were over the 160 character limit for SMS messages (shown as max SMS). We also encoded the evaluation data using the LEC compression algorithm [7]. The four horizontal lines in Figure 6 compare the mean length of DARA messages with the entropy mean, the LEC mean and the maximum SMS message length. The mean message length for the DARA algorithm (shown as DARA mean of 104.7) is close to the entropy lower limit (shown as entropy mean of 96). The mean compression ratio for LEC (shown as LEC mean of 133.1) is 28.6 characters longer than the mean DARA. The standard deviation of DARA was 11.1 characters and for LEC is 7.7 characters. Both algorithms have similar memory and execution requirements.

According to their data sheets the TELIT GC 864-QUAD typically draws $<$420 mA for GPRS transmissions while the ATmega 1284p draws 0.4 mA in active mode. So, in practice, no more than 1% of the energy is used for sensing and compression, and 99% for sending the SMS message. Without compression, our system could send at most 2 sets of readings in a single SMS message, using a format such as Table 2 (left). This would require 12 SMS messages to be sent per day. With compression, a whole day of readings can be compressed into a single SMS message. Given the dominating energy cost of transmissions, saving 11 SMS messages in each a sequence of 12 packages sent to the receiver, is worth much more than the best possible energy-aware optimisations of a compression algorithm.

Since we assumed that there is no message loss in our experimental SMS scenario, we could avoid spending a few extra bits for sending codebook versions and repeatedly sending codebook changes after at most every N messages, although we still transmitted an anchor with each message. Within the evaluation period, there were no codebook changes required. So, for this case, there was no overhead for codebook maintenance. Where codebook maintenance is required, there were, on average, 55 characters available per message, and so the codebook maintenance information together with the compressed sensed data would easily fit into a single SMS message. In summary, DARA comfortably meets its compression requirements for this application, having a mean message length close to Shannon entropy limit and being 28 characters per message shorter than LEC, which has the best performance of existing sensor network compression algorithms. Further, LEC transmits only encoded differences with no anchors, and so it relies on the assumption that no messages will be lost in transmission.

6 Conclusions

This paper presents the design and evaluation of a generic algorithm, DARA, for efficient compression of sensor network data. Previous studies have focussed on resource-awareness, optimising existing algorithms to respect resource constraints. Our approach is both data-aware and resource-aware. The DARA algorithm defines a data-aware mapping from raw sensor readings to an equivalent form that maximises compression. Once this is done, standard Huffman coding methods can be used, albeit with some modifications for resource constrained nodes. The DARA algorithm is general purpose: it is suitable for heterogeneous or homogeneous sensor data streams; it can be trained on different types of sensor data; and it can be configured to achieve higher or lower compression ratios as demanded by different applications. The algorithms is also adaptive: coding parameters are changed on-the-fly when necessary. In comparison with existing compression algorithms for sensor networks, the advantage of our algorithm is that it can easily integrate expert knowledge about the optimal data representation, which leads to better compression ratios whilst still respecting the resource constraints of sensor nodes. In particular, practical use of DARA is demonstrated by an SMS application that requires blocks of sensor readings to be compressed to one quarter of their original size.

There are a number of possible directions for future work. In considering an SMS application, the problem of lost messages was insignificant. However, if the compression algorithm were to be used in a less reliable communication environment, then further study of methods for recovering lost messages and managing lost state information for adaptive coding would be needed. Further investigation of the effect of adaptive coding in long running applications is needed, and comparison with existing data stream-mining methods and new types of sensor data streams.

Acknowledgements. This work was partially supported by the Go8-DAAD Australia-Germany Joint Research Co-Operation Scheme under grant DAAD10505012.

References

1. Barr, K.C., Asanović, K.: Energy-aware lossless data compression. ACM Trans. Comput. Syst. 24(3), 250–291 (2006)
2. Bell, T., Witten, I.H., Cleary, J.G.: Modeling for text compression. ACM Comput. Surv. 21(4), 557–591 (1989)
3. bzip2 (2012),`http://bzip.org` (retrieved January 2012)
4. Calgary Corpus Geophysical data (1987), `www.data-compression.info/Corpora/CalgaryCorpus/` (retrieved June 2012)
5. Capo-Chichi, E.P., Guyennet, H., Friedt, J.-M.: K-RLE: A new data compression algorithm for wireless sensor network. In: Third International Conference on Sensor Technologies and Applications, SENSORCOMM 2009, pp. 502–507 (June 2009)
6. Guitton, A., Trigoni, N., Helmer, S.: Fault-Tolerant Compression Algorithms for Delay-Sensitive Sensor Networks with Unreliable Links. In: Nikoletseas, S.E., Chlebus, B.S., Johnson, D.B., Krishnamachari, B. (eds.) DCOSS 2008. LNCS, vol. 5067, pp. 190–203. Springer, Heidelberg (2008)
7. Marcelloni, F., Vecchio, M.: An efficient lossless compression algorithm for tiny nodes of monitoring wireless sensor networks. Computer Journal 52(8), 969–987 (2009)
8. Reinhardt, A., Christin, D., Hollick, M., Schmitt, J., Mogre, P.S., Steinmetz, R.: Trimming the Tree: Tailoring Adaptive Huffman Coding to Wireless Sensor Networks. In: Silva, J.S., Krishnamachari, B., Boavida, F. (eds.) EWSN 2010. LNCS, vol. 5970, pp. 33–48. Springer, Heidelberg (2010)
9. Reinhardt, A., Christin, D., Hollick, M., Steinmetz, R.: On the energy efficiency of lossless data compression in wireless sensor networks. In: IEEE 34th Conference on Local Computer Networks. LCN 2009, pp. 873–880 (October 2009)
10. Sadler, C.M., Martonosi, M.: Data compression algorithms for energy-constrained devices in delay tolerant networks. In: Proceedings of the 4th International Conference on Embedded Networked Sensor Systems, SenSys 2006, pp. 265–278. ACM, New York (2006)
11. Salmon, D.: Huffman Coding. In: A Concise Introduction to Data Compression, ch. 2, Springer, Heidelberg (2008)
12. Schoellhammer, T., Greenstein, B., Osterweil, E., Wimbrow, M., Estrin, D.: Lightweight temporal compression of microclimate datasets. In: 29th Annual IEEE International Conference on Local Computer Networks, pp. 516–524 (November 2004)
13. Verma, N., Zappi, P., Rosing, T.: Latent variables based data estimation for sensing applications. In: 2011 Seventh International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP 2011), Adelaide, Australia, pp. 335–340 (December 2011)
14. WebSense: Sensor Network Viewer (2011), `wsn.csse.uwa.edu.au/` (retrieved June 2012)