Data Compression Algorithms for Energy-Constrained Devices in Delay Tolerant Networks

Christopher M. Sadler and Margaret Martonosi

Department of Electrical Engineering
Princeton University
{csadler, mrm}@princeton.edu

Abstract

Sensor networks are fundamentally constrained by the difficulty and energy expense of delivering information from sensors to sink. Our work has focused on garnering additional significant energy improvements by devising computationally-efficient lossless compression algorithms on the source node. These reduce the amount of data that must be passed through the network and to the sink, and thus have energy benefits that are multiplicative with the number of hops the data travels through the network.

Currently, if sensor system designers want to compress acquired data, they must either develop application-specific compression algorithms or use off-the-shelf algorithms not designed for resource-constrained sensor nodes. This paper discusses the design issues involved with implementing, adapting, and customizing compression algorithms specifically geared for sensor nodes. While developing Sensor LZW (S-LZW) and some simple, but effective, variations to this algorithm, we show how different amounts of compression can lead to energy savings on both the compressing node and throughout the network and that the savings depends heavily on the radio hardware. To validate and evaluate our work, we apply it to datasets from several different real-world deployments and show that our approaches can reduce energy consumption by up to a factor of 4.5X across the network.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless communication; E.4 [**Coding and Information Theory**]: Data compaction and compression

General Terms

Algorithms, Performance, Reliability

Keywords

Data Compression, Energy Efficient Communications, Mobile Ad Hoc Networks, Wireless Sensor Networks

SenSys'06, November 1–3, 2006, Boulder, Colorado, USA.

1 Introduction

In both mobile and stationary sensor networks, collecting and interpreting useful data is the fundamental goal, and this goal is typically achieved by percolating data back to a base station where larger-scale calculations or coordination are most feasible. The typical sensor node design includes sensors as needed for the application, a small processor, and a radio whose range and power characteristics match the connectivity needs and energy constraints of the particular design space targeted.

Energy is a primary constraint in the design of sensor networks. This fundamental energy constraint further limits everything from data sensing rates and link bandwidth, to node size and weight. In most cases, the radio is the main energy consumer of the system. For example, in a ZebraNet sensing node which includes a power-hungry GPS sensor, radio transmit and receive energy still represents around 60% of the total energy budget [42]. That percentage can be even higher in a Mica2 node [12], which features lower-energy sensors and a shorter-range CC1000 radio.

Given the high proportion of energy spent on communications, it is natural to try to reduce radio energy. For this reason, this paper examines data compression techniques geared to improving sensor network energy. Tailoring data compression approaches to sensor nodes requires a few key shifts in mindset, as we describe below.

First, standard compression algorithms are aimed at saving storage, not energy. As a result, compression ratio is their fundamental metric. In this paper, we focus on energy savings as the primary metric instead. We consider both the local energy tradeoffs for a single data-producing node performing the compression, as well as the downstream energy impacts for intermediate routing nodes who benefit from handling less data.

Second, the potential for both local as well as downstream energy savings suggests that more aggressive compute-intensive compression strategies are worth exploring. Figure 1 highlights this with an energy-oriented comparison. For three commonly-used sensor radios, we illustrate how many compute cycles can be performed for the same energy usage as transmitting a single byte over the radio. (This graph is based on real measurements of the three radios, summarized in Table 2 later in the paper, and the TI MSP430 [35] processor, which has emerged as a popular sensor processor choice.) The takeaway message from this graph is that if a compression computation can result in even just one

byte of data reduction, then it is worth spending between roughly four thousand (Chipcon CC2420) to two million (MaxStream XTend) cycles of computation to accomplish that compression. Clearly, compression algorithms with some degree of intricacy are worth exploring! Note further that these calculations only consider local energy consumption at the compressing node; downstream energy savings can further amortize the time/energy expense of compression.

Third, the memory size constraints of current and near-future sensor processors require a re-examination of memory usage in compression calculations. While sensor processors are showing increasing capabilities in each technology generation, they still typically offer less than 50 kB of code memory and even less data RAM. Thus, compression algorithms originally geared for desktops or servers must be restructured to reduce the code size footprint and dynamic memory usage.

This paper evaluates lossless data compression options and proposes novel approaches tailored to the unique tradeoffs and constraints of sensors. We explore lossless compression in this work because it is applicable to a wider variety of applications, datasets, and users than lossy compression, and we leave lossy compression for future work. The contributions of this paper include:

- We propose and evaluate a family of basic lossless compression algorithms tailored to static and mobile sensor networks and applicable to a wide range of applications. Evaluated by real-system measurements on a range of sensor data sets, we find these algorithms can cut energy consumption by 2X or more.
- We discuss additional steps for transforming or preconditioning the data that can further reduce energy. For "structured" sensor data, these transforms can further reduce energy consumption by roughly 3X compared to the initial baseline requirements.
- We evaluate the downstream energy benefits of compression on all subsequent intermediary nodes that receive and forward data, both for links with 100% transmission reliability as well as for cases where some packet loss and retransmission occurs. Overall, we find that for realistic reliability scenarios, even sensor networks with very-low-energy communication can benefit from time spent on data compression by amortizing the time/energy/compute costs over the network.

The remainder of this paper is structured as follows. Section 2 discusses related work. Section 3 then presents our measurement methodology in preparation for the results presented in Section 4 on the baseline compression algorithm, methods for tailoring it to sensing, and a variant designed for sensor data. Section 5 discusses methods for performing preconditioning or data transformation steps to further improve compression for certain structured data streams. Section 6 quantifies compression benefits under different reliability and replication scenarios, Section 7 offers a summary and discussion, and Section 8 concludes the paper.

2 Related Work

Compression is a well-established research field, but sensor networks present a new design space with new design

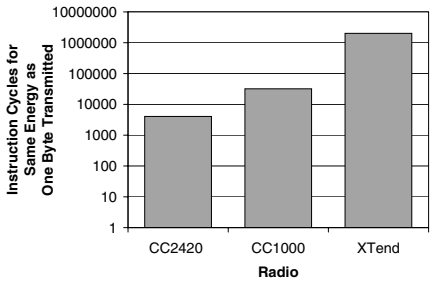

Figure 1. Number of TI MSP430F1611 compute cycles that can be performed for the same amount of energy as transmitting one byte over three radios.

tradeoffs and metrics. In particular, the small code and data memories, and the primary focus on energy, call for new approaches. In more traditional arenas, a large variety of compression algorithms have been proposed including string-based techniques [19][32][43], entropy-based techniques [11][15][40], and others.

Within the sensor networks community, some more tailored data compression studies do already exist. One group of prior work focuses on exploiting high spatial correlation in data from fixed sensors in dense networks [10][14][26]. Our work, in contrast, is more general in its ability to handle data from a range of systems, particularly those that are either sparse or mobile, for which spatial data correlation is uncommon. Another group of prior sensor-oriented compression work has focused on data-centric routing and aggregation [1][4][16][24][25][29][34]. Again, this research often exploits high spatio-temporal correlation in the data streams from dense, fixed sensor networks. While our proposals include some transforms for preconditioning structured data, they do not assume any correlation based on spatial relationships of sensor nodes or on temporality between packets of data. As such, they are more general and are better equipped to handle the sorts of low-correlation situations common to mobile sensor networks. They can also be applied in concert with some of these other techniques.

This paper focuses on LZW [39], which has received significant research attention including so-called "embedded" versions [17][18][23]. While these approaches do typically use less memory than their counterpart algorithms aimed at high-end machines, they are still not well-suited for sensor applications. In some cases, their memory usage still exceeds the tens of kilobytes typical for sensor nodes. In other cases, their assumptions about static dictionary entries are not appropriate for sensor nodes where data is expected to vary significantly over the duration of a deployment.

In one of the papers most closely related to our own, Barr and Asanović specifically aim to trade computation energy for compression savings [5]. There are, however, several key differences between their work and ours. First, the goal of their work is to re-evaluate existing, off-the-shelf compression algorithms for mobile devices using energy as the key metric as opposed to compression ratio. Our work, on the other hand, proposes unique compression approaches (and combinations of approaches) that are distinct variants of prior techniques and are specifically tailored to the energy and memory constraints of sensor nodes. Second, our work

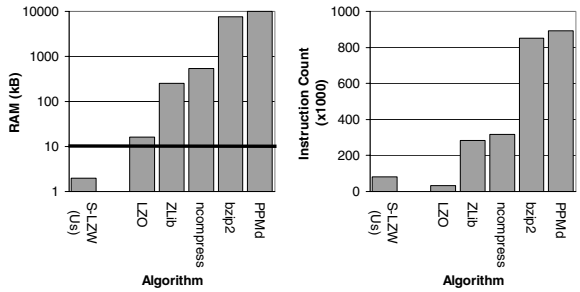

Figure 2. Brief comparison of the compression algorithms profiled in [5] and S-LZW (our algorithm). Left: The compiler default memory requirement of the algorithm's core table. The line represents the amount of RAM on our MSP430 microcontroller. Right: The average number of instructions (given by the SimIt StrongArm simulator [27]) required to compress 528B of data from the four benchmarks we profile in this paper.

ties energy directly to specific radios from different points across the sensor design space; this allows us to make concrete energy tradeoffs for a range of current and future sensor nodes.

Additionally, the striking differences between their target platform (233 MHz StrongArm with 32 MB RAM, a 16 kB cache, and an external file system) and ours (4 MHz fixed-point microcontroller with 10 kB RAM and 48 kB ROM) require us to take a different approach to compression in general. Our work studies data streams generated by actual sensor deployments that we break up into independent 528B blocks to ease transmission requirements, while their work focuses on more generic data streams from text files or the web which they compress in 1 MB blocks.

Figure 2 shows the instruction counts and the default core memory usage of the algorithms evaluated in [5] versus our LZW implementation for sensor nodes, S-LZW. None of the algorithms they evaluated are directly transferable to our sensor node because *they were not designed for sensor nodes*. Four of the five algorithms have memory and/or processing requirements that far exceed what our microcontroller can provide. The exception is LZO, which they show consumes the least of amount of energy to compress and send 1 MB of data of all the algorithms they evaluated. The developers of LZO implemented a version specifically for embedded systems such as ours called miniLZO. We will briefly evaluate miniLZO in Section 4.4 and show that for sensor nodes, our algorithm has a superior energy profile.

3 Methodology

This section describes the datasets we use for our tests, the hardware on which the tests are performed, and the way in which we evaluate energy consumption.

3.1 Datasets

To examine how compression algorithms perform in a real-world deployment, we test them against datasets from the SensorScope (SS) [30], Great Duck Island (GDI) [33], and ZebraNet (ZNet) [42] deployments, plus a popular benchmark, the geophysical dataset from the Calgary Corpus (Calgeo) [6]. These datasets represent a large portion of the sensor network design space: indoor, stationary networks; outdoor, stationary networks; and outdoor, mobile networks

Dataset	Comp. Ratio (first 528B)	Comp. Ratio (all)	Compressibility
SensorScope [30]	3.69	4.58	High
Great Duck Island [33]	1.59	1.94	Medium
ZebraNet [42]	1.38	1.96	Medium-Low
Calgary Corpus Geo [6]	1.11	1.49	Low

Table 1. Compression profile for experimental datasets based on gzip.

respectively. Calgeo, which is comprised of seismic data, is a commonly-used compression benchmark. We use the data from the beginning of the real-world datasets; for example, when we use 2 Flash pages (528 bytes) of data, that is the first 528 bytes in the set. For Calgeo, the first 200 bytes have a different profile from the rest of the dataset so we remove those and use the next 528 bytes. We have also experimented with data from elsewhere in the datasets and found that it did not qualitatively change our conclusions.

These three real-world networks are Delay Tolerant Networks [13], which we define to be networks that may intentionally buffer data before transmitting it. In practice, buffering may be done out of necessity, as in networks with long or frequent disconnections, or as part of an energy conservation mechanism, such as the work described in this paper. We estimate that it would have taken these networks between one and four hours to collect 528B.

The ZNet dataset contains a combination of GPS data and debugging data, which consists of mostly voltage data and information on the performance of the network. The GPS data has already been pre-compressed once with an application specific algorithm based on offsets of past positions. As a result, the data actually fed into the compressor is more variant over short time intervals than that of the other datasets.

To give a snapshot of the variability in each dataset, we compress them with the popular LZ77-based compression algorithm gzip on our PC, and the results are displayed in Table 1. These numbers provide initial insight into how well our compression algorithms will perform on these datasets.

Our datasets exhibit temporal locality, but they do not display the spatial correlation that other researchers in the sensor network community are currently exploring; the data from each node is important so we focus on lossless compression in this work.

3.2 Experimental Platform

Although the actual sensor hardware used in the actual deployments varies, we run all of our experiments on a TI MSP430x1611 microcontroller (10 kB RAM, 48 kB on-chip flash memory) running at 4 MHz, to allow for a fair comparison [35]. It consumes just 5.2 mW, and is used in both the Tmote Sky [21] and ZebraNet [42] sensor nodes. This microcontroller consumes about one-third as much energy per cycle as the 8-bit Atmel ATmega128L microcontroller [3] at the same frequency and voltage, and it has 2.5 times more

RAM. Moore's Law improvements of this type mean that increasingly capable processors will become commonplace.

Nodes in the original SS and GDI deployments transmitted data as soon as it was generated; however, since these networks are delay tolerant, we assume here that sensor readings are stored in Flash until there is enough data to compress. Only sensor readings are stored and compressed; additional information such as node ID and sample rate can be added to the beginning of the transmission and would not be transmitted with every set of readings.

Our board features a 0.5 MB Atmel Flash module [2] that communicates with the microcontroller at a rate of 1 Mbps. The memory is broken into independent pages of 264 bytes, and this module consumes 512 μJ to write a page to nonvolatile memory, 66 μJ to read a page, and a negligible amount of energy when powered down. Data must be written in multiples of the page size (we cannot save energy by writing back data in smaller quantities), so we prefer to work with data in multiples of this size.

For our tests, data to be compressed is stored in Flash ahead of time and is read out a page at a time during the test. It is then compressed and written back to Flash in pieces as the output buffer fills and upon completion of the compression. Also, we assume that the pages have been pre-erased and we do not perform flash erasures here.

Since we are using Flash memory, the compression times are longer than if we only worked with RAM. This method makes sense, however, because we can compress data blocks of varying sizes without having to store the entire input and output streams in the limited memory available. Each algorithm therefore requires at least 528 bytes of RAM for one page input and output buffers.

To measure the compression time, we connected a LeCroy Waverunner 6030 oscilloscope to an unused pin on the microcontroller. We drive a pin immediately before calling the compression algorithm, and release the pin immediately upon returning from the algorithm.

3.3 Quantifying Energy

We quantify the energy savings compression achieves both at the compressing node and throughout the network. We evaluate our work with three different radios of different ranges and power levels.

To exemplify short, medium, and long range radios, we use the Chipcon CC2420 [9], the Chipcon CC1000 [8], and the MaxStream XTend (at 500 mW transmit power) [20] radios respectively. They were chosen because they are built into the Tmote Sky, Mica2 [12], and ZebraNet nodes so they cover a large portion of the design space. The CC2420 is more likely to be used in stationary systems, while the XTend is more appropriate for mobile systems since the mobility creates a need for increased range.

Table 2 shows the listed range and measured power data for each radio. To measure power, we connected a low ohmage, current sense resistor in series with the load to be measured and used a DAQ to measure the voltage drop over that resistor. Simple calculations yield the current flowing through the load and from that, the power consumption. We measured the power consumed by the XTend radio, the microcontroller, and the Flash directly by cutting electrical traces on a ZebraNet node that was running a basic communication program on top of ZebraNet firmware. For the CC2420 radio, we measured the system power of a Tmote Sky node running a simple TinyOS-based communication program and subtracted that from the system power measured when the radio was off (since we were unable to attach a resistor in series with the radio directly). We obtained the power values for the CC1000 from [31] and the throughput from comments in the TinyOS source code [36].

Radio	Range	TX Power	RX Power	Baud Rate
Chipcon CC2420	125 m	46.7 mW at 3V	50.9 mW at 3V	70,677
Chipcon CC1000	300 m	64.5 mW at 3V [31]	21 mW at 3V [31]	12,364 [36]
MaxStream XTend	15 km	2.43 W at 5V	444.5 mW at 5V	7,394

Table 2. Radio profiles obtained via measurements.

The bandwidth listed for the XTend radio is for every packet after the first in a string of packets (streaming mode). However, every time the packet switches from receive mode to transmit mode, the time required to transmit the first packet increases by over 40% as the radios synchronize. Our models account for this parameter.

We assume that data is transmitted in packets of 64 bytes, with 10 bytes of header. These numbers are from the ZebraNet deployment, and they fit in reasonably with the state of the art. The last packet in the chain only sends the header plus the data remaining to be sent; for example, if we have sent 270 bytes of data out of 300 total, the last packet will be 40 bytes.

4 Tailoring Compression Techniques for Sensor Nodes

4.1 Overview and LZW Basics

LZW is a lossless, dictionary-based algorithm that builds its dictionary as data is read in from the input stream [39][41]. It is a good fit for sensor networks because the dictionary structure allows it to adapt to fit any input and take advantage of repetition in the data.

At the start, the dictionary is initialized to 256 entries, one for each eight-bit character. It then executes the process illustrated in Figure 3. An example is shown in Figure 4.

This algorithm has no transmission overhead and is computationally simple. Since both the sender and the receiver have the initial dictionary and all new dictionary entries are created based on existing dictionary entries, the recipient can recreate the dictionary on the fly as data is received.

To decode a dictionary entry, however, the decoder must have received all previous entries in the block. Unfortunately, sensor nodes never deliver 100% of their data to the source. For this reason, our algorithm separates the data stream into small, independent blocks so that if a packet is lost it only affects the data that follows it in its own block (Figure 5). Later in this subsection, we will experimentally determine the appropriate size of these data blocks with a bias towards smaller blocks that minimize the amount of data that can be lost due to a packet drop. For now, our experi-

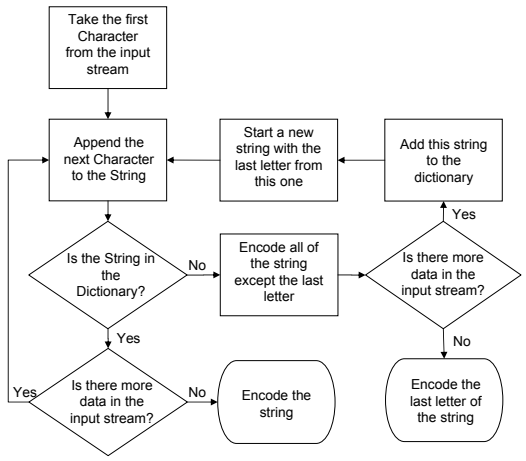

Figure 3. Flow chart of LZW compression.

Input Stream: AAAABAAABCC

Encoded String	Output Stream	New Dictionary Entry
A	65	256 - AA
AA	65 256	257 - AAA
A	65 256 65	258 - AB
B	65 256 65 66	259 - BA
AAA	65 256 65 66 257	260 - AAAB
B	65 256 65 66 257 66	261 - BC
C	65 256 65 66 257 66 67	262 - CC
C	65 256 65 66 257 66 67 67	

Figure 4. Example of LZW compression.

ments assume perfect transmissions. We explore imperfect transmissions in Section 6.1.

To adapt the algorithm to sensor nodes, the dictionary can be stored as a hash table with the base entries being the initial dictionary. Strings can then be represented as branches from that table so they are easy to locate. Further, we can make the memory requirement manageable by keeping the dictionary small. An implementation with a 512 entry dictionary requires 2618 bytes (plus four bytes for each additional entry) of RAM and 1262 bytes of ROM. This means that the algorithm requires at least a quarter of the RAM available on this chip; however, this is still feasible for current systems and over time the amount of RAM available is likely to steadily increase.

4.2 LZW for Sensor Nodes (*S-LZW*)

To adapt LZW to a sensor node, we must balance three major inter-related points: the dictionary size, the size of the data to be compressed, and the protocol to follow when the dictionary fills. First, our memory constraints require that we keep our dictionary size as small as possible. Additionally, as mentioned, we want to compress and decompress relatively small, independent blocks of data.

Both of these issues factor into our third decision regarding what to do when the dictionary fills. The two options are to freeze the dictionary and use it as-is to compress the remainder of the data in the block, or to reset it and start from scratch. Sensor data is changing over time so after the dictionary is full, the entries in the dictionary may not be a good representation of the data being compressed at the end of the block. However, that is typically not a problem if the data block is small since the dictionary will not fill until the block has almost been completely compressed.

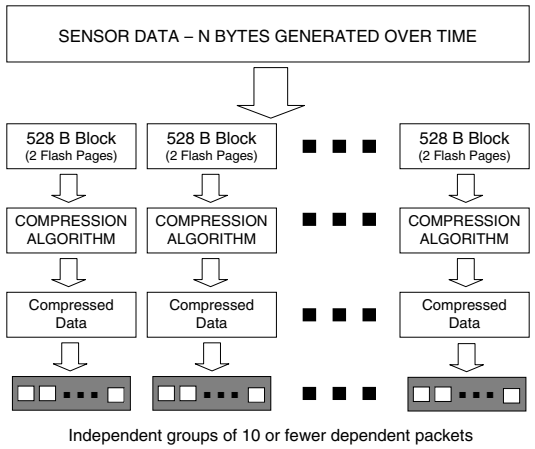

Figure 5. Data is separated into individual blocks before being compressed and divided into packets. Each packet in a shaded block cannot be decompressed without the packets that proceed it, but packets in different shaded blocks are independent of one another.

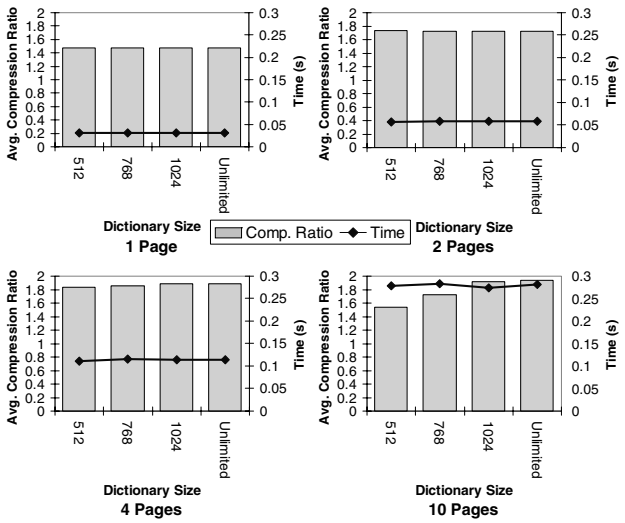

Figure 6. Average compression ratios and execution times for our four datasets using LZW to compress a given number of Flash pages of data.

We first determine the appropriate dictionary size for a sensor node and the amount of data to collect before compressing it. While we determine these values, we will freeze the dictionary once it fills.

Dictionary Size: Our experiments focus on dictionaries of 512, 768, 1024, and for comparison, an unlimited number of entries. Strings are encoded in 9 bits while the dictionary has less than 512 entries, 10 bits while it has less than 1024 entries, etc. Figure 6 shows how the average compression ratio and execution time of our algorithm changes with dictionary size when compressing data in blocks of 1, 2, 4, and 10 Flash pages. With small data blocks, there is almost no difference between the dictionary sizes.

With large amounts of data, however, the compression ratio increases with dictionary size. For example, in compressing 4 pages of ZNet data, there is over 12% penalty for using a 512 entry dictionary instead of a 1024 entry dictio-

nary. In compressing 10 pages of SS, ZNet, and Calgeo data, the size penalties are 56.7%, 14.3%, and 5.7% respectively. The penalty occurs because with data blocks this large, the dictionary fills and is fixed long before the entire block has been compressed. Meanwhile, the sensor data is changing over time, so by the time the algorithm reaches the end of the block, the entries in the dictionary are no longer a good representation of the data being compressed. Given these results and our memory constraints, S-LZW will use a 512 entry dictionary from this point forward. As a direct result, we will focus our experiments on data sizes of 4 pages or less.

Data size: Next, we must decide how much data to compress at once. We evaluate this algorithm for data block sizes of one, two, and four Flash pages of data by compressing two Flash pages in either one block or two individual one-page blocks and four Flash pages in either one block or two individual two-page blocks.

The compression ratio benefits significantly from compressing data in a block of two pages rather than in two separate one page blocks. For the SS, GDI, ZNet, and Calgeo datasets, there are improvements of 25.5%, 12.9%, 4.6%, and 3.3% respectively. The large improvements for SS and GDI are because they are more predictable over short periods than ZNet and Calgeo. Therefore, dictionary entries created for the first page are more applicable to the second.

When compressing data in a block of four pages rather than in two blocks of two pages, though, the benefits decrease significantly. For the SS, GDI, ZNet, and Calgeo datasets, there are improvements of 12.1%, 3.9%, -8.2%, and 2.0% respectively. It is better to compress the ZNet data in two blocks of two pages because the dictionary fills before the compression is complete and older entries are not applicable to the data at the end of the block.

The compute times for compression have a similar pattern. The time required to compress two pages of data with the SS dataset is 24.5% less than the time required to compress two single pages of data. The GDI, ZNet, and Calgeo datasets have more modest reductions of 1.7%, 5.3%, and 1.1% respectively. Savings come mostly from avoiding the initialization time required to create and start building the dictionary. When compressing four pages versus two sets of two pages, though, the benefits level out. There is a 1.1%-3.3% decrease when compressing four pages in a block depending on the dataset.

Overall, the time required to accumulate the data to compress must be balanced with the compression ratio and execution time. Since only the SS dataset sees a significant increase in compression ratio when compressing data in four page blocks instead of two individual two page blocks, S-LZW will compress data in blocks of 528 bytes (2 Flash pages).

Dealing with a Full Dictionary: Finally, we must revisit the protocol to follow when the dictionary fills. Since we are using a data block size of 528 bytes, the dictionary does not fill until the block has almost been completely compressed. Compared to the reset protocol, the fixed protocol completes an average of 2.4% faster and the compression ratio is an average of 2.4% better. The fixed protocol is slightly faster because it does not have to reset the dictionary. Its compression ratio is slightly better because the dictionary is still relevant for a short period of time after it fills up. Thus, S-LZW will use a fixed protocol.

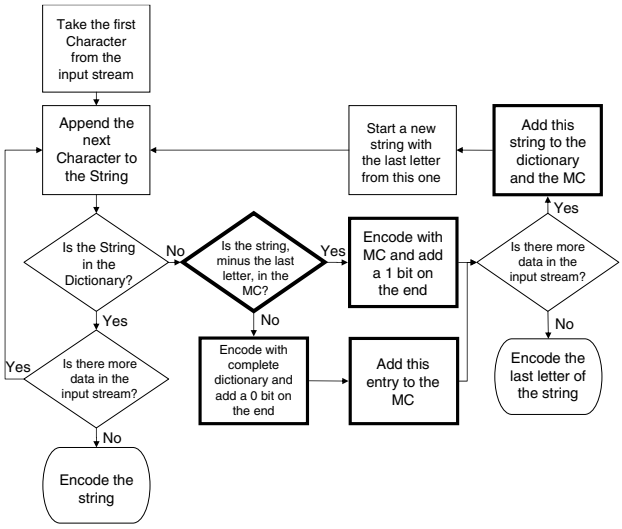

Figure 7. Flow chart of S-LZW-MC.

4.3 S-LZW with Mini–Cache (*S-LZW-MC*)

S-LZW is a good general compression algorithm, but it does not take specific advantage of sensor data characteristics. Sensor data tends to be repetitive over short intervals. Even data that changes drastically over time tends to adhere to this pattern because of short sample intervals designed to catch those drastic changes in the data. To optimize for these patterns, we propose a new idea for compression: adding a so-called mini-cache to the dictionary.

The mini-cache is a hash-indexed dictionary of size N, where N is a power of 2, that stores recently used and created dictionary entries. For our hash index, we use the last four bits of the dictionary entry. This index is both simple and effective; recently created entries are given dictionary numbers in sequential order, so they will wrap around the mini-cache before overwriting each other. S-LZW with a 32-entry mini-cache (*S-LZW-MC32*) requires 2687B of RAM and 1566B of code (69B of RAM and 304B of ROM more than S-LZW).

As shown in the flow chart in Figure 7, this algorithm follows the S-LZW algorithm until the encoding stage. Once it has identified the dictionary entry, it searches the mini-cache for the entry.

If the entry is in the mini-cache, the algorithm encodes it with the mini-cache entry number and appends a '1' bit onto the end yielding an entry $(\log_2 N) + 1$ bits long. If not, it encodes the entry with the standard dictionary entry number, appends a '0' bit onto the end, and then adds this entry to the mini-cache. A high mini-cache hit rate will allow the algorithm to make up for the one-bit penalty on items not in the mini-cache.

Once the entry is encoded, a new dictionary entry is created just like in the S-LZW scheme and that entry is also added directly to the mini-cache. This leaves the most recently used and created entries in the mini-cache. An example of this process is shown in Figure 8.

Input Stream: AAAABAAABCC

Encoded String	New Output	New Dict. Entry	Mini-Cache Changes	Total Bits: LZW	Total Bits: Mini-Cache
A	65, 0	256 - AA	0-256, 1-65	9	10
AA	0, 1	257 - AAA	1-257	18	15
A	65, 0	258 - AB	1-65, 2-258	27	25
B	66, 0	259 - BA	2-66, 3-259	36	35
AAA	257, 0	260 - AAAB	1-257, 4-260	45	45
B	2, 1	261 - BC	5, 261	54	50
C	67, 0	262 - CC	3-67, 6-262	63	60
C	3, 1			72	65

Figure 8. Example of S-LZW-MC.

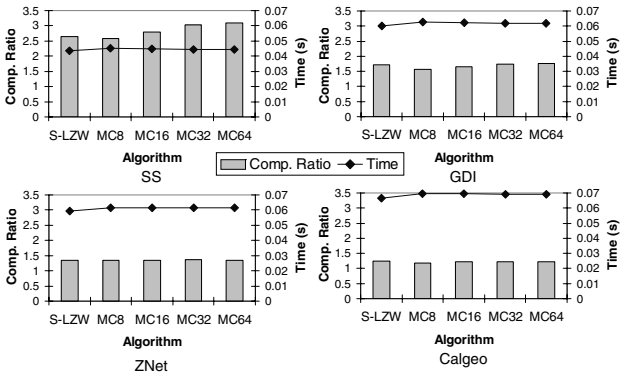

Figure 9. Compression ratio and execution time for our baseline algorithm (S-LZW) and its mini-cache variations (S-LZW-MC8, MC16, MC32, MC64).

The most important design decision to make is the size of the mini-cache. Intuitively, entries in smaller caches are encoded in fewer bits but have a lower hit rate. We evaluate our algorithm for powers of two between 8 and 64 entries and our results are shown in Figure 9.

There is less than a 4.5% computation overhead for this algorithm in all cases. The 32 and 64 entry dictionaries work best here, and we see substantial gains in compression ratio with the SS data compared to S-LZW (12.9% and 14.7% for the 32 and 64 entry dictionaries respectively). The other three datasets experience minimal improvements, but we will explore ways to improve these results in the next section.

In addition to this simple hashing approach, we have also experimented with other indexing algorithms such as LRU. We omit them from this paper partly due to space constraints and partly because they offered little benefit in compression, but took much longer to compute.

4.4 Results: Local Energy Evaluation

Most prior compression work has used compression ratio as the primary metric. In sensor networks, however, energy is what matters. The next two subsections quantify the degree to which even small size savings can lead to substantial energy savings.

Local energy savings refers to the energy saved immediately on the compressing (originating) node versus just transmitting the data without any compression at all. Section 4.5 will evaluate the downstream effects on other nodes.

For the base case without compression, local energy consumption is just the energy required to transmit the whole block of data. For a node with compression, it is the sum of the energy required to transmit the compressed data, the energy required by the microcontroller to compress that data, and the energy consumed by the Flash module.

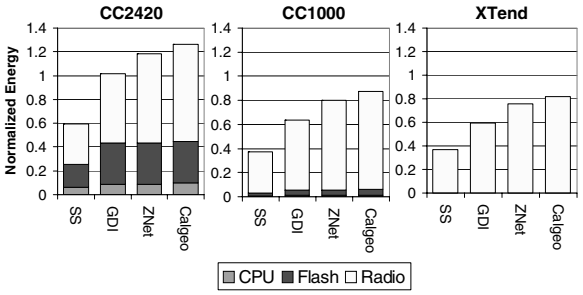

Figure 10. S-LZW-MC32 energy consumption with our short, medium, and long range radios normalized to transmitting without compression (values < 1 represent local energy savings).

Our local energy evaluations in this paper account for the following effects:

- Microcontroller compression energy: This consists of the measured microcontroller power multiplied by the measured compression time.
- Flash read and write energy.
- Radio transmission energy: This consists of the radio transmit power plus the microcontroller power multiplied by the transmission time.
- Radio start-up and shut-down energy: The start up energy is derived from our power measurements and the datasheet recommendations for the delay (1ms, 1.5ms, and 100ms for the CC2420, CC1000, and XTend radios respectively). When the radio is turned off, the power is cut so shut-down is close to instantaneous.

Figure 10 shows the energy requirements of compressing data with S-LZW-MC32 and transmitting it, normalized to that of just sending the data in an uncompressed form. Energy is broken up into microcontroller energy, flash energy, and transmission energy.

In almost all cases, S-LZW offers energy savings. Our medium and long range radios reduce energy by an average factor of 1.7X across all four datasets over transmitting without compression, including an energy reduction of a factor of over 2.6X on the SS dataset.

With our short range radio, the CC2420, however, there is a slight average increase in net energy consumption. This is due to the cost of writing the final result back to our Flash module; Flash energy accounts for an average of over 30% of the total energy consumed. Using this radio but not the Flash, we would reduce energy by 2.4X with the SS dataset and an average of 1.6X over all four datasets. Even with the Flash, though, compression offers further downstream benefits that we will explore in the next subsection.

miniLZO: miniLZO [19] is an LZ77 derivative designed with the processing, memory, and code size requirements of embedded systems in mind. It compresses files by taking strings of three or more characters and replacing strings that are repeated with pointers further back in the file. It is derived directly from the LZO algorithm evaluated by [5],

which we discussed in Section 2. We evaluate it here for completeness.

We found that miniLZO requires just 56% of the time to run to completion as S-LZW. However, the compressed files are 23.5% larger than those compressed by S-LZW. In fact, it could not compress data from Calgeo at all because it is hard to find repeating runs of three or more characters in that dataset.

In terms of energy, we found that on average, S-LZW is between 9.6% (CC2420) and 17.5% (XTend) more energy-efficient than miniLZO and that miniLZO's advantage in compression time does not outweigh its poor compression ratio. Based on these results, we do not consider miniLZO further in this work.

4.5 Results: Downstream Energy

Compression conserves a large amount of energy in downstream nodes. Downstream energy savings refers to the energy saved by relay nodes when sending data in its compressed form rather than its original form. Since these nodes do not have to do any compression work, they will *always* save energy assuming the compressed data size is less than the uncompressed data size.

Our downstream energy calculations in this paper account for the following effects:

- Radio transmission energy.
- Radio start-up and shut-down energy.
- Radio receive energy: This consists of the time required to receive all the data multiplied by the power required to receive it.

From the local and downstream energies, we calculate total network energy, which is our metric for measuring the overall network benefits of compression. A packet that travels a single hop is considered to have traveled directly from the compressing node to the basestation, so it does not consume downstream energy. (We do not consider basestation energy in this paper.)

To evaluate the effects of network scaling, we use the S-LZW-MC32 compression algorithm since it performs well for all of the datasets and all of the radios. Figure 11 shows that as the network size and average hop count increases, overall network energy savings increase linearly. Additionally, this figure shows that the system saves more energy per hop as radio range increases.

Overall, we can conclude this amount of compression is almost always beneficial to the network *even if there is a small penalty on the compressing node*. This is especially important for relay nodes close to the basestation that are required to pass on a great number of packets; compressing with S-LZW-MC32 reduces energy consumption with the short and medium range radios at each intermediate hop by up to 2.9X with the SS dataset and an average of 1.8X overall. These numbers are actually higher than those achieved locally with the long-range radio, and this is due to the difference in start-up energy between the radios and the energy required to receive the data on downstream nodes.

Decompression: Although the datasets we analyze do not display enough data correlation to aggregate at intermediary nodes, our methods can provide energy savings in systems that can benefit from aggregation as long as intermediary nodes can decompress the data. We implemented S-LZW decompression on our system and measured that the decompression times were on average about 60% of the compression times, which is less than the amount of energy required to send 34B with the short-range radio. Additionally, the process was completed in about 57.1% of the time required to send the data over that radio at the measured baud rate, so the designer may be able to reduce the energy burden by decompressing the data while it is being received and the CPU is already active. Since our datasets cannot benefit from this, however, we do not explore the issue further in this work.

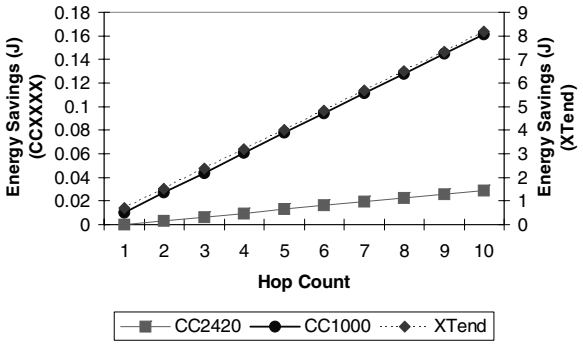

Figure 11. Energy savings vs. hop count using GDI data and S-LZW-MC32 for all three radios. CC2420 and CC1000 are against the left axis, XTend is against the right.

Flash Effects: Downstream energy savings are maximized when intermediary nodes do not have to store relayed data in non-volatile memory. However, in delay tolerant networks, long disconnections can occur which would necessitate storing data in Flash. On systems with longer range radios, using non-volatile storage increases the energy requirement to about the same as the local energy requirement plus the cost of receiving the compressed data from the prior hop (we do not incur computational energy costs here, but those are very low in comparison). Alternatively, systems with short range radios see a slightly larger net benefit because the compression energy is non-negligible.

Storing data on intermediary nodes is better than the alternatives, which are to drop packets once the buffer fills (what most systems currently do), or to receive and store the uncompressed data. Compared to receiving and storing the data uncompressed, using S-LZW-MC32 saves an average of 1.7X across the three radios. The SS dataset garners an additional benefit because data is compressed to the point where it fits into one flash page; other variants of this algorithm presented later in this work can achieve this benefit as well.

5 Data Transforms for Small Memory Compression

The S-LZW-MC algorithm conserved energy by taking advantage of characteristic locality patterns in the sensor data. One way that we can possibly create additional patterns in blocks of data is to transform them in a reversible manner. In this section, we will evaluate transforms designed to reorder the data in a way to make it easier to compress.

5.1 Burrows–Wheeler Transform

The Burrows–Wheeler Transform [7] (BWT) is a reversible transform central to the bzip2 compression algorithm. By itself, the BWT does not shrink the size of the data; its purpose is to make the data easier to compress. Here, we explore its use as a precondition step before S-LZW.

The data transformation is accomplished by taking the last column of a sorted matrix of shifted copies of our input stream. Using the last column allows the decoder to reconstruct the original data stream, but leaves the data organized in a way that contains runs of characters.

In more capable computers, this transform is often used as the first of a three part process. Typically, after the BWT is performed, the data is run through a Move-to-Front (MTF) encoder and then through an entropy encoder. However, MTF obscures higher order repetition that can be exploited by S-LZW. Our preliminary tests show that for all four datasets we can achieve better compression ratios without it, so we do not consider it in this paper.

Our implementation is a variant of one presented in [22], modified to work with our 16-bit processor and Flash module rather than a PC. It has a seven byte (1.33% of 528 bytes) overhead to enable the decompression algorithm to reverse the transform.

One disadvantage is that the BWT requires that we allocate RAM for the entire input and output streams and for a large buffer to perform the required sorts. However, we can significantly reduce the memory requirement by sharing RAM with the S-LZW back end. Implementing the BWT in this fashion requires only 281 extra bytes of memory over S-LZW, most of which is to store the entire stream to be compressed with S-LZW, and this adjustment makes implementation reasonable for sensor nodes.

5.1.1 Compression Time and Size Results

Figure 12 plots the compression ratio and execution time for the four datasets using BWT-based algorithms. BWT is a helpful precondition that dovetails well with our mini-cache approach. This transform creates runs in the data, which, in turn, improve the mini-cache hit rate. The best compression ratio improvements are found with a 16 entry dictionary, where the SS, GDI, and ZNet datasets see size improvements of 39.7%, 11.0%, and 9.8% respectively over S-LZW without the BWT.

These size improvements come with a computational cost, however, due to the large number of string compares required to perform the necessary sorts. We use an iterative quicksort algorithm, with the required stack stored in the shared memory block, but our measurements show that the *S-LZW-MC-BWT* algorithms still require 0.32 to 0.35 seconds to compress 528 bytes of data.

5.1.2 Local Energy Savings

Since BWT offers promising size improvements at considerable computational cost, the local and downstream (5.1.3) energy effects are the final arbiter of its utility.

The local energy costs for each of the three radios, normalized to the energy required to send data without compression, are shown in Figure 13. For each bar, the bottom portion represents the energy consumed by the microcontroller, the middle portion represents flash energy, and the top portion represents transmission energy.

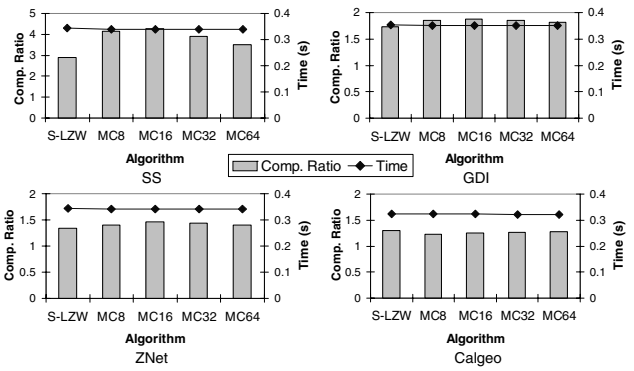

Figure 12. Compression ratio and execution time for BWT-based compression algorithms (Note that the y-scale on the upper-left hand graph differs from the others.)

With the long-range radio, we achieve significant local energy savings with the S-LZW-MC-BWT algorithms. Even with the long computation time, the computation energy is so small compared to the transmission energy that it cannot even be seen on the graph. With this radio, the S-LZW-MC16-BWT algorithm reduces energy consumption by an average factor of 1.96X.

Moving to the mid-range radio, we still see some energy benefits. The SS dataset has close to a 3.0X improvement and the energy costs when compressing the other three datasets are within 4.1% of those incurred by the S-LZW-MC32 algorithm.

With our short-range radio, the computational costs of the BWT implementations are nearly half of the cost of just transmitting the data without compressing it at all. When combined with the costs of Flash storage, from a *local* energy standpoint it does not seem worthwhile to use BWT-based algorithms to compress the data for the CC2420.

5.1.3 Downstream Energy Savings

Downstream savings can make aggressive compression worthwhile even when it does not save energy for the local (compressing) node. On the short range radio, compressing with S-LZW-MC16-BWT rather than with S-LZW-MC32 saves an average of about 10.7% more energy at each hop (2.1X overall). By the third hop, however, we save energy compressing data from all four datasets. This means that in large networks, nodes with short range radios far from the basestation begin to take on the same characteristics as nodes with long range radios; it is worth spending a great amount of energy processing data to avoid having to transmit it.

For the mid and long-range radios, both of which saved energy locally, compressing with the S-LZW-MC16-BWT algorithm saves an average of 9.9% more energy at each hop than S-LZW-MC32 and an average factor of 2.04X overall.

5.2 Replacing BWT in Structured Datasets

To this point, we have assumed that we know very little about the data we are trying to compress. However, if we can identify a simple, reversible way to reorder the data, we could surpass the gains obtained with BWT at a reduced energy cost.

For the SS, GDI, and Calgeo datasets, we know the exact number and size of the data readings the nodes will generate every time they activate. Based on this, we will refer to

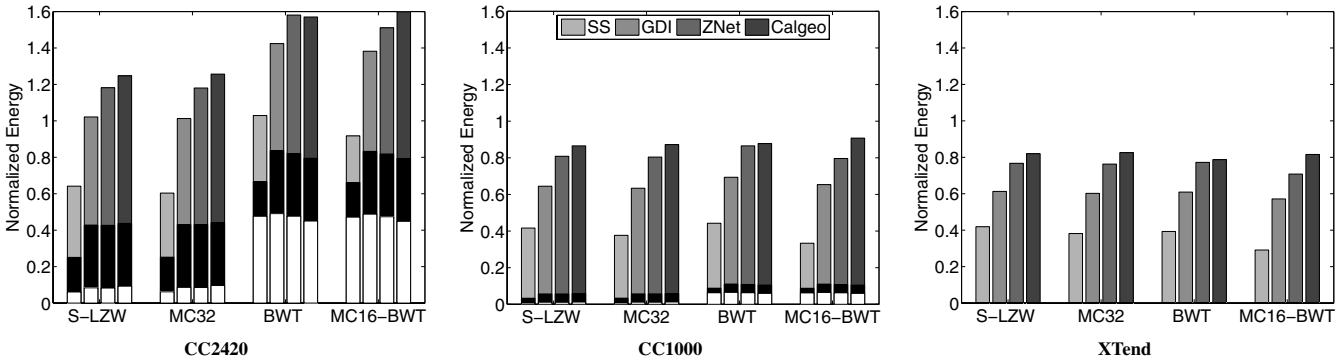

Figure 13. Local energy normalized against no compression for S-LZW and three variants (S-LZW-MC32, S-LZW-BWT, and S-LZW-MC16-BWT). Values < 1 represent local energy savings. The bottom portion is computation energy, the middle portion is Flash energy, and the top portion is transmission energy.

them as "structured" datasets. We also know that some of the readings are identical over the short intervals and for the 16-bit readings, even if two readings from the same sensor vary moderately, they are likely to have the same most significant byte.

In these cases, we propose that the system form a matrix of readings in which each row is a set of readings taken at a given time. We know the number and size of the readings that are acquired when the node is activated, so each row will be the same length. We keep filling the matrix until the total number of elements is as close to 528 bytes as possible without breaking up a row. Finally, we transpose that matrix and compress the data in that form. This Structured Transpose (*ST*) is easy to execute and to reverse, and it groups data in a way that is easier to compress than data transformed by BWT.

This implementation has a similar memory penalty to BWT, requiring 266 bytes more RAM than S-LZW (assuming a 528 byte block of data) and 16 more bytes of code. In both cases, the entire input stream must be read into RAM for the transformation, and again we share memory with the S-LZW back-end to alleviate the problem. Adding the ST to the S-LZW-MC schemes yields similar penalties.

In this subsection, we also discuss the Run Length Encoding (RLE) compression algorithm. We did not consider RLE earlier because it does a uniformly poor job of compressing the raw sensor data; that data is unlikely to have long runs of identical characters. However, the ST creates these runs, so this algorithm is an option worth exploring here.

RLE is a lossless, computationally simple algorithm that replaces runs of characters with character counts. Any runs of two or more identical characters are replaced by two instances of the character plus the length of the rest of the run. Single characters are left alone. *RLE-ST* requires 772 bytes of program code and 814 bytes of RAM with the transpose, mostly for the page-sized input and output buffers.

The ZNet dataset has no defined structure since the size and order of the entries varies, so we do not consider it in this subsection.

5.2.1 Compression Size and Time Results

The primary advantage to using the ST as opposed to the BWT is that there are no string comparisons, so it is extremely fast, and therefore, much more likely to conserve energy on systems with short range radios. Figure 14 shows the compression ratios and execution times for the RLE-ST, S-LZW-ST, and S-LZW-MC-ST algorithms.

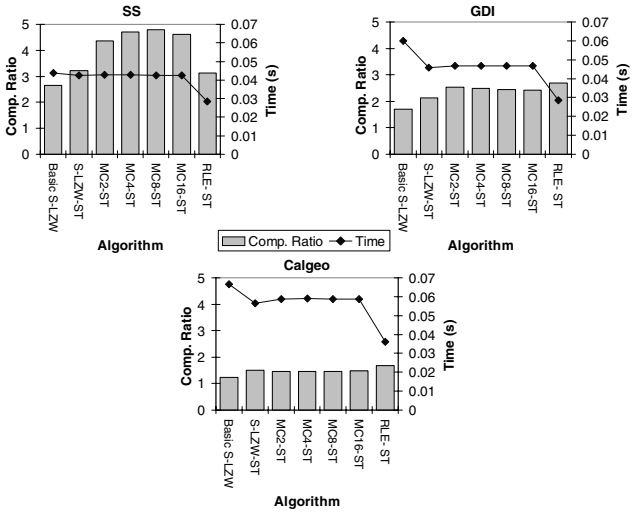

Figure 14. Compression ratio and execution time for compression algorithms with the Structured Transpose (ST). Basic S-LZW does not perform the ST.

Running the S-LZW-ST based algorithms across the three datasets, the SS, Calgeo, and GDI datasets compress an average of 2.3%, 12.3%, 22.3% faster than S-LZW, respectively. This algorithm proves very effective with the mini-cache since smaller mini-caches work well, so every hit saves more bits. S-LZW-MC8-ST compresses best, with compression ratios of 4.8, 2.4, and 1.5 for the SS, GDI, and Calgeo datasets respectively.

RLE-ST proves to be the fastest algorithm we explore in this subsection by far. Its compression time is 33.4%-39.2% less than for S-LZW-MC8-ST. Additionally, it does a good job of compressing the data with ratios of 3.1, 2.7, and 1.7 for the SS, GDI, and Calgeo datasets respectively.

5.2.2 Local Energy Savings

Figure 15 shows that large energy savings can be obtained by compressing with the ST. All of the schemes yield energy savings on all of the radios for all three datasets even when compared to already-good S-LZW compression.

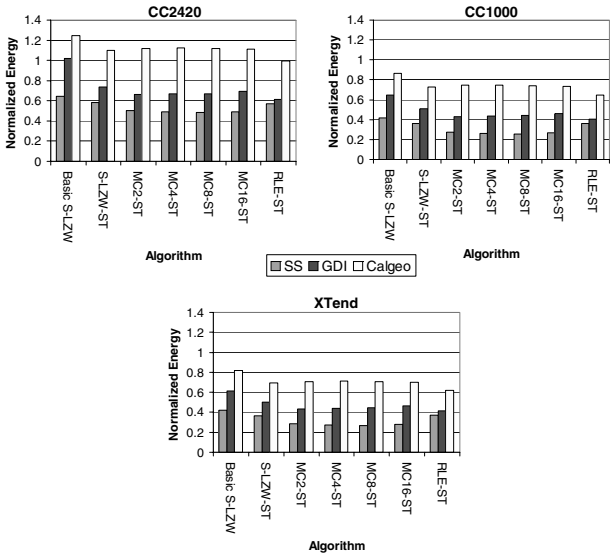

Figure 15. Local energy normalized against no compression (values < 1 represent local energy savings). All algorithms except Basic S-LZW perform the ST.

On all of the radios, the SS dataset benefits from the S-LZW-MC8-ST algorithm, due to the 45% improvement in compression ratio over S-LZW without the ST. On our mid-range radio, this translates to a 3.9X energy reduction. Overall with this radio, we reduce energy by an average factor of 2.8X.

For both the GDI and Calgeo datasets, RLE-ST performs best because it is the fastest and has the best compression ratio. With our mid-range radio, this reduces energy consumption by a factor of 2.5X for the GDI data and 1.5X for the Calgeo data. Additionally, RLE-ST achieves *local* energy savings with all three datasets on the short range radios; it is the only algorithm we have explored so far for which this is true.

The primary advantage of using the ST schemes over distributed sensor node compression algorithms that require previous knowledge of the data is that this ST can be done locally on the original node. That allows it to work in heterogeneous networks and in networks with little to no data correlation. Additionally, local and downstream energy savings are achieved earlier, since the data does not need to be passed to a compressing node.

5.2.3 Downstream Energy Savings

With the mid-range radios, compressing the datasets with S-LZW-MC8-ST is an average of 28.2% more energy-efficient than using S-LZW-MC32 (no ST). This translates to an overall energy reduction of 2.8X, including a reduction of 4.5X for the SS dataset.

The RLE-ST algorithm provides significant energy reduction as well, and is preferred for the short range radio in smaller networks since it achieves local energy savings. Under these conditions we cut energy consumption on each downstream node by an average factor of 2.4X compared to transmitting without compression, an improvement of 21.2% over S-LZW-MC32 (no ST).

6 Effect of Reliability and Replication on Compression Benefits

To this point, we have assumed that packet transmissions are perfect. However, in real networks, transmission are not perfect and this can have a significant energy impact. Compressing sensor data can have additional benefits as the number of retransmissions required increases. In this section we explore the benefits of compression in both networks with variable reliabilities and in opportunistic, mobile systems that use packet replication to increase the probability of successfully transmitting packets to the sink.

6.1 The Effects of Reliability

To quantify the energy savings of compression in an unreliable network, we use our short range radio, GDI data, and S-LZW-MC32 compression as an example, and trends observed are representative of other datasets and compression algorithms. We define reliability as the success rate of packet transmissions.

Since we are only compressing and sending data in small blocks, we use an out-of-order transmission protocol for this work. In our model, the sender transmits the whole compressed block, which is at most 10 packets. Packets are modeled as being dropped independently at the prescribed rate based on a Monte-Carlo model. The receiver then sends a 32B acknowledgement that tells the sender which packets it received correctly and the sender retransmits what is missing. If the transmission is not complete and the receiver does not start receiving data immediately, it assumes that the acknowledgement was lost and retransmits it. This process iterates until the receiver has successfully received and acknowledged the whole block. For each packet drop rate, we take the average energy consumption from 1000 iterations of the model. In the style of delay tolerant networks [13], we also assume that retransmissions are only required over the hop in which the packet was dropped rather than having to restart from the source node. Given the costs of retransmissions, this is a good way to implement reliability in sensor networks.

Figure 16 shows how the network energy savings increase as the success rate decreases when using the short-range CC2420 radio. The increase in savings is due directly to the increased number of retransmissions necessary in the scenario. In this figure, the energy savings at each individual point are measured against sending the data without compression at the prescribed hop count and packet success rate. At a given reliability, energy savings increases linearly with hop count. At 100% reliability, this algorithm/data/radio combination did not see local energy savings, but if reliability drops to just 90% we do conserve energy locally. On a two-hop network, we see an overall network energy savings immediately.

Real deployments have exhibited comparably poor reliabilities. For example, Szewczyk et al. report that the GDI deployment successfully delivered over 70% of its packets to the sink in its single hop network, but one quarter of the nodes in this setup delivered less than 35% [33]. The multi-hop network delivered just 28%, although a number of these losses were deterministic rather than as a result of node to node drops. Most nodes were within five hops of the base-station. Schmid et al. report that the SS deployment fared

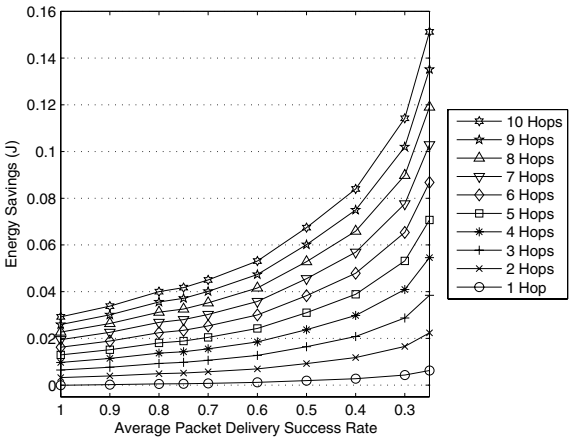

Figure 16. Network energy saved when sending a block of data with compression at different reliabilities versus sending without compression (higher is better). Based on GDI data compressed with S-LZW-MC32 on a CC2420 radio.

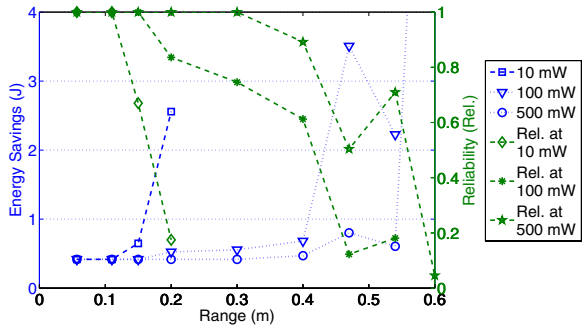

Figure 17. Local energy savings vs. transmission distance for ZNet data compressed with S-LZW-MC32 and transmitted with the XTend radio at different power levels. Dots (against left axis): Energy saved versus sending without compression. Dashes (against right axis): Reliabilities at different ranges.

slightly better, but the delivery rate quickly dropped below 50% as the average depth increased past two hops [30]. This means that compression will be worthwhile across all setups.

The ZebraNet deployment featured mobile nodes, and reliability varied with range. Figure 17 depicts the local energy savings for ZNet data compressed with S-LZW-MC32 versus transmission distance on our long range radio at transmission powers of 10 mW, 100 mW, and 500 mW (distances measured in an urban environment). Even at short ranges, the reliability is highly variant. As reliability drops, energy consumption spikes. An interesting side note is that as the transmission distance approaches the maximum range for a given transmission power, increasing the transmission power can actually save energy because of the improved reliability. In our experiences with actual deployments, the observed reliability was worse and even more variant over range because of the poor transmission conditions such as the lack of guaranteed line of sight, node movement during transmissions, disruption of the signal by the animal's body, and damage to the antenna over time.

Given the high packet loss rates observed in all three of these deployments, we can conclude that our compression

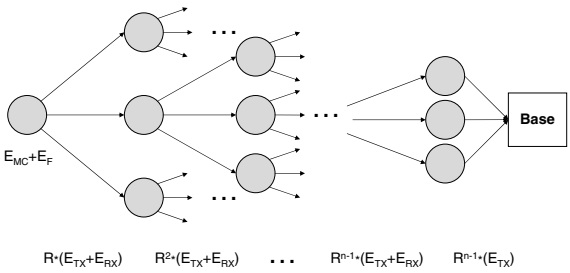

Figure 18. A diagram of a replication scheme with $R = 3$.

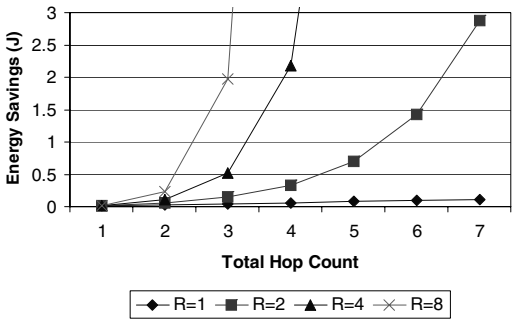

Figure 19. Energy saved at different replication totals for GDI data compressed with S-LZW-MC32 protocol versus sending without compression on a CC1000 radio as hop count increases.

algorithms would significantly increase the system's life expectancy.

6.2 The Effects of Replication

Replication is a transmission strategy primarily used in opportunistic, mobile systems that improves the speed and probability of successfully delivering packets to the basestation by sending them to more than one node. It is central to the epidemic routing scheme [37][38]. If you assume that the first $n-1$ nodes along the path each replicate the packet R times and that the last hop only transmits back to the basestation, the energy consumed by an n hop network is given by:

$$E_1 = E_{MC}+E_F+E_{TX}$$

$$E_n = E_{MC}+E_F+\left(\sum_{i=1}^{n-1} R^i\right)*(E_{TX}+E_{RX})+R^{n-1}(E_{TX})$$

$$\text{for } n \geq 2$$

E_{MC} and E_F are the microcontroller and flash energies, and E_{TX} and E_{RX} are the transmit and receive energies (factoring in the radio start-up and shut-down energies). The last set of R^{n-1} transmissions signify passing the data back to the basestation. This equation is illustrated in Figure 18.

Figure 19 profiles the potential energy savings in systems with replication. It presents the GDI data on a mid-range radio, but is representative of our other datasets. Each point on the x-axis represents the total hop count, n, so $n-1$ nodes are allowed to replicate packets. The curves represent replication factors, R, of 1, 2, 4, and 8.

When $R = 1$, this system is the same as any normal packet propagation scheme; each node sends to exactly one neighbor and the last hop is to the basestation. As the number of

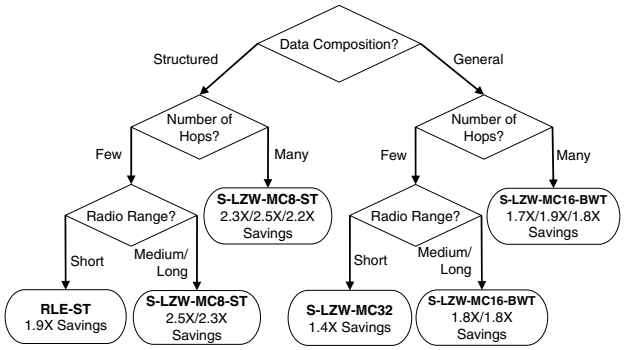

Figure 20. Summary of the best algorithms in terms of network energy savings in a network with few hops and with many hops (energy savings based on networks with 2 and 5 hops, respectively). Note that in networks with many hops, the algorithm choice is irrespective of radio range and the savings listed are for Short/Medium/Long-Range radios.

replications per hop increases, however, the energy savings increases exponentially with hop count. These data are independent of Section 6.1 and thus assume 100% reliability.

7 Summary and Discussion

In summary, we achieve local energy savings in almost every situation we have explored in this paper. Additionally, we have shown the significant downstream energy savings achieved through compression. We have also shown that it is often beneficial to the network overall to compress data even if there is a local energy loss.

We find that a family of compression algorithms is more appropriate for sensor networks rather than just using one, and that the appropriate compression algorithm depends on the hardware, the size of the network, and a priori knowledge of the data's structure. Figure 20 illustrates our findings. The numbers represent the factor by which we reduce total network energy (both local and downstream energy) in smaller (2-hop) and larger (5-hop) networks.

In small networks with the short-range radio, the computational requirement of the compression algorithm factors into our decision and warrants a different algorithm than we use for longer-range radios in the same scenario. For structured data, RLE-ST offers the best energy and it saves local energy for all three datasets. On the other hand, S-LZW-MC8-ST is more versatile and saves energy if there is at least one downstream node. For unstructured data, S-LZW-MC32 (no transform) also saves network energy overall in a two-hop network, and the compression saves energy at nodes close to the sink that relay lots of packets, so we recommend this algorithm even on short-range radios. As the network grows, though, the compression choice becomes independent of the radio as the savings accumulate at each hop.

Although the energy profile of the Flash causes some of our algorithms lose energy locally when using the short-range radio, Flash technology is improving and Moore's Law dictates that the energy profile will continue to improve. Our experiments are conducted with an older Flash module that has been used in a number of sensor nodes and deployments. As the technology improves, however, Flash energy will become much less significant and store and forward data communications will become even more attractive. This will improve the benefits of compression in delay tolerant networks.

For networks with the XTend radio, compression garners a huge net energy savings: 843 mJ per hop with S-LZW-MC16-BWT and 1.1 J per hop with S-LZW-MC8-ST. The percentage by which the energy is reduced is less than that of the medium-range radio, however, because there is a large energy cost incurred to activate this radio and begin a transmission, and this cost is incurred at each hop regardless of whether or not we compress data.

Navigating through a large group of algorithms can be daunting to a system designer, so to help analyze our findings we also compare our results against oracle (ideal) energy savings. This value is derived by running all of our compression algorithms across all of the 528B blocks that compose each dataset and using the most energy efficient algorithm to compress each block. We found that for each dataset/radio combination, at least one compression algorithm garnered energy savings within about 3% of the Oracle. Therefore, unless the one of the characteristic properties of the system changes, a single algorithm will likely perform well throughout the duration of the deployment. Additionally, for mobile networks and networks with highly variant hop counts, the need for longer range radios simplifies the decision since with high energy radios the choice is the same for both large and small networks.

Future Work: We would like to implement lossy compression algorithms that can be applied to a wide variety of sensor applications with minimal effort. Such algorithms must minimize computation while maintaining a specified accuracy, and this may be achievable with predictor-based algorithms or with simple techniques derived from standard audio and visual compressors.

We would then like to apply both lossless and lossy compression algorithms to data from groups of nodes in dense networks. If the network can determine the amount of correlation in the data from those nodes at a minimal energy cost, we may achieve network energy savings by aggregating and compressing it with lossy techniques at an intermediate hop. Additionally, we may save energy by compressing data on the source node with the lossless techniques described in this paper and then decompressing it at the aggregating node.

8 Conclusion

This paper explores the design issues and energy implications of implementing generalized, lossless compression algorithms on delay tolerant sensor networks. We introduce S-LZW, a novel LZW variant that exploits characteristic patterns of sensor data to reduce energy consumption by more than 1.5X as well as further data transforms that can take advantage of the structure of the data to decrease network energy consumption on average by over a factor of 2.5X.

These algorithms are relevant to a wide variety of sensor hardware implementations and applications; we demonstrate this by applying our algorithms to data from three different real-world deployments and examining their effects on three radios of different ranges and power levels. As radio energy increases, so do the benefits of compression, since each byte saved becomes increasingly significant. Through these evaluations, we show how compression accumulates substantial energy savings at the compressing node and as data is passed

through every intermediary node between the source and the sink.

Additionally, we show that compression has a greater impact in networks with poor reliability. Published results of real-world deployments have shown that the packet delivery rate in multi-hop networks can easily be well below 50% and that in mobile networks, reliability decreases sharply as the transmission distance increases. Under these conditions, network energy savings more than double due to the need for retransmissions or packet replication.

Overall, this work offers contributions at two levels. First, we quantify the high leverage that compression has on energy issues for sensor networks. Building on that, we have proposed and evaluated new algorithms and approaches tailored to sensor network scenarios. This work offers a family of solutions that span the broad design space of sensor networks.

Acknowledgements

We would like to thank Robert Szewczyk (UC Berkeley) and Thomas Schmid (EPFL) for their help with the GDI and SensorScope datasets respectively; Ken Barr and Kriste Asanovic (MIT) for helping us re-evaluate their work with respect to our platform; Pei Zhang (Princeton) for helping to set up the sensor hardware and measurements; and our shepherd for this paper, Sergio Servetto (Cornell). This work was supported in part by NSF ITR program (CCR 0205214).

9 References

[1] T. Arici, B. Gedik, Y. Altunbasak, and L. Liu. PINCO: a Pipelined In-Network Compression Scheme for Data Collection in Wireless Sensor Networks. In *IEEE Intl. Conf. on Computer Communications and Networks*, 2003.

[2] ATMEL. AT45DB041B, 4M bit, 2.7-Volt Only Serial-Interface Flash with Two 264-Byte SRAM Buffers data sheet. `http://www.atmel.com/`, June 2003.

[3] ATMEL. 8-bit AVR Microcontroller with 128K Bytes In-System Programmable Flash Datasheet. `http://www.atmel.com/`, 2004.

[4] S. J. Baek, G. de Veciana, and X. Su. Minimizing Energy Consumption in Large-Scale Sensor Networks Through Distributed Data Compression and Hierarchical Aggregation. *IEEE Journal on Selected Areas in Communications*, 2004.

[5] K. Barr and K. Asanović. Energy Aware Lossless Data Compression. In *Proc. of the ACM Conf. on Mobile Systems, Applications, and Services (MobiSys)*, May 2003.

[6] T. C. Bell, J. G. Cleary, and I. H. Witten. *Text Compression*. Prentice Hall, 1990.

[7] M. Burrows and D. J. Wheeler. A Block-sorting Lossless Data Compression Algorithm. *Digital Systems Research Center Research Report*, 124, 1994.

[8] Chipcon AS. Chipcon SmartRF CC1000 Datasheet rev. 2.3. `http://www.chipcon.com/`, Aug. 2005.

[9] Chipcon AS. Chipcon SmartRF CC2420 Datasheet rev. 1.3. `http://www.chipcon.com/`, Oct. 2005.

[10] J. Chou, D. Petrovic, and K. Ramchandran. A Distributed and Adaptive Signal Processing Approach to Reducing Energy Consumption in Sensor Networks. In *Proc. INFOCOM*, 2003.

[11] J. G. Cleary and I. H. Witten. Data Compression using Adaptive Coding and Partial String Matching. *IEEE Transactions on Communications*, COM-32(4):396–402, April 1984.

[12] Crossbow Technology, Inc. Mica2 Datasheet. `http://www.xbow.com/`, 2005.

[13] K. Fall. A Delay Tolerant Network Architecture for Challenged Internets. In *Proc. of the Special Interest Group on Data Communications Conf. (SIGCOMM)*, 2003.

[14] D. Ganesan, B. Greenstein, D. Perelyuskiy, D. Estrin, and J. Heidermann. An Evaluation of Multi-Resolution Storage for Sensor Networks. In *First International Conference on Embedded Networked Sensor Systems*, 2003.

[15] D. A. Huffman. A Method for the Construction of Minimum-Redundancy Codes. In *Proceedings of the I. R. E.*, 1952.

[16] C. Intanagonwiwat, R. Govindan, and D. 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 (MOBICOM)*, Aug. 2000.

[17] H. Lekatsas, J. Henkel, and V. Jakkula. Design of an One-cycle Decompression Hardware for Performance Increase in Embedded Systems. In *Proc. Design Automation Conference (DAC)*, 2002.

[18] C. H. Lin, Y. Xie, and W. Wolf. LZW-Based Code Compression for VLIW Embedded Systems. In *Proc. of the Design, Automation and Test in Europe Conf. (DATE)*, 2004.

[19] Markus Oberhumer. LZO Real-Time Data Compression Library. `http://www.oberhumer.com/opensource/lzo/`, Oct. 2005.

[20] Maxstream, Inc. XTend OEM RF Module: Product Manual v1.2.4. `http://www.maxstream.net/`, Oct. 2005.

[21] Moteiv Corp. Tmote Sky: Low power Wireless Sensor Module Datasheet. `http://www.moteiv.com/`, Mar. 2005.

[22] M. Nelson. Data Compression with the Burrows-Wheeler Transform. *Dr. Dobbs Journal*, Sept. 1996.

[23] E. Netto, R. Azevedo, P. Centoducatte, and G. Araujo. Mixed Static/Dynamic Profiling for Dictionary Based Code Compression. In *Proc. Intl. Symposium on System-on-Chip*, 2003.

[24] S. Pattem, B. Krishnamachari, and R. Govindan. The Impact of Spatial Correlation on Routing with Compression in Wireless Sensor Networks. In *Proc. of the Symp. on Information Processing in Sensor Networks (IPSN)*, 2004.

[25] D. Petrovic, R. C. Shah, K. Ramchandran, and J. Rabaey. Data Funneling: Routing with Aggregation and Compression for Wireless Sensor Networks. In *SNPA Workshop, ICC 2003 Intl. Conf. on Communications*, 2003.

[26] S. S. Pradhan, J. Kusuma, and K. Ramchandran. Distributed Compression in a Dense Microsensor Network. *IEEE Signal Processing Magazine*, pages 51–60, March 2002.

[27] W. Qin. SimIt-ARM. `http://simit-arm.sourceforge.net/`, Mar. 2003.

[28] B. M. Sadler. Fundamentals of Energy-Constrained Sensor Network Systems. *IEEE Aerospace and Electronics Systems Magazine, Tutorial Supplement*, 2005.

[29] A. Scaglione and S. D. Servetto. On the Interdependence of Routing and Data Compression in Multi-Hop Sensor Networks. In *Proc. of the Intl. Conf. on Mobile Computing and Networking (MOBICOM)*, 2002.

[30] T. Schmid, H. Dubois-Ferriere, and M. Vetteri. SensorScope: Experiences with a Wireless Building Monitoring Sensor Network. In *Proc. of the Workshop on Real-World Wireless Sensor Networks (RealWSN)*, 2005.

[31] V. Shnayder, M. Hempstead, B. Chen, G. Werner-Allen, and M. Welsh. Simulating the Power Consumption of Large-Scale Sensor Network Applications. In *Proc. of the ACM Conf. on Embedded Networked Sensor Systems (SenSys)*, Nov. 2004.

[32] J. A. Storer and T. G. Szymanski. Data Compression via Textual Substitution. *Journal of the ACM*, 29:928–951, 1982.

[33] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler. An Analysis of a Large Scale Habitat Monitoring Application. In *Proc. of the ACM Conf. on Embedded Networked Sensor Systems (SenSys)*, Nov. 2004.

[34] C. Tang, C. S. Raghavendra, and V. K. Prasanna. Power Aware Coding for Spatio-Temporally Correlated Wireless Sensor Data. In *IEEE Intl. Conf. on Mobile Ad-Hoc and Sensor Systems*, 2004.

[35] Texas Instruments. MSP430x161x Mixed Signal Microcontroller Datasheet. `http://www.ti.com/`, Mar. 2005.

[36] TinyOS Community Forum. TinyOS. `http://www.tinyos.net/`.

[37] A. Vahdat and D. Becker. Epidemic routing for partially connected ad hoc networks. In *Technical Report CS-200006, Duke University*, Apr. 2000.

[38] Y. Wang, S. Jain, M. Martonosi, and K. Fall. Erasure-Based Routing for Opportunistic Networks. In *Proc. of the Special Interest Group on Data Communications Conf. (SIGCOMM)*, Aug. 2005.

[39] T. A. Welch. A Technique for High-Performance Data Compression. *IEEE Computer*, 17(6):8–19, June 1984.

[40] I. Witten, R. Neal, and J. Cleary. Arithmetic Coding for Data Compression. *Commun. ACM*, 30:520–540, June 1987.

[41] WxWiki. Z File Format. `http://www.wxwidgets.org/wiki/index.php/Development:_Z_File_Format`.

[42] 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.

[43] J. Ziv and A. Lempel. A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.