Hardware Assisted Compression in Wireless Sensor Networks

Navraj Chohan
Dept. of Computer Engineering and Computer Science
University of California, Santa Barbara
nchohan@cs.ucsb.edu

June 2007

Abstract

A wireless sensor network (WSN) is comprised of resource constrained devices called motes. Motes are battery operated and must conserve as much power as possible. Power usage for each mote is made up of processing data, sensing the surroundings, and sending and receiving packets. The bulk of power usage for standard WSN applications lies in the radio where receiving and sending packets are necessary to communicate to a base station (as much as 80%) [5]. This communication is generally greater than the range of any single mote, and thus multi-hop routing must take place. The multi-hop routing furthers the need for reduction of packets because each additional hop adds to the amount of traffic generated. By compressing sense data, which each mote collects, the number of packets can be reduced. Only lossless compression is taken into consideration. This paper proposes a solution to reduce the consumption of power using reconfigurable hardware specially programmed to do low power compression. A mote would offload bulk data to a hardware attachment and receive the compressed version of the data. The compressed data can be stored in flash or sent over the network to the sink node. Additional work will go into the trade-off factor between compression to non-compression. Compression should only occur if the amount of power taken to compress X amount of bytes and transfer Y amount of bytes is less than just transferring X amount of bytes. Moreover, seeing the gains from doing offloading of compression versus doing it on-chip must be taken into consideration as well. Additional hardware is not free, so cost for the power savings plays a factor in deciding how compression takes place. Moreover, there are also limitations based on the kind of reconfigurable hardware. This paper addresses these issues.

1 Introduction/Background

Wireless sensor networks (WSNs) are comprised of resource constrained embedded devices called motes. Motes are placed in an ad-hoc fashion to sense the environment and report back their readings to a base station or sink node. Sink nodes are generally out of the range of many of the motes. Motes must use peer-to-peer links to cope with the lack of direct communication. A network is built in a distributed fashion which results in a minimum spanning tree as seen in Figure 1. This paper aims to take advantage of the communication model of a sensor network by minimizing the amount of bits transmitted through compression.

Motes are constrained in resources. The primary resource is their battery supply. The battery supply limits the lifetime of a mote as well as sets the scale of their form factor. For example, the MICAz mote runs on two AA batteries, providing 3V and 2000mAH (a brief summary of power terminology is given in Section 2). If an application takes on average 25mA to run (100% duty cycle), the lifetime of the motes is approximately three and a third days [8]. Having a low duty cycle is key to extending the lifetime of

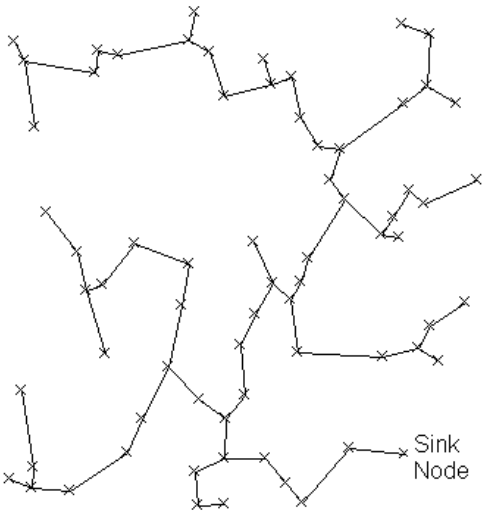

Figure 1: A minimum spanning tree with one sink node.

the mote. Normal operation of the microprocessor, radio, and flash take in the order of miliamps to operate. When putting these components to sleep the cost is on the order of microamps. The key to any application is only having the necessary components active and the rest asleep.

The popular motes in today's WSNs are built by Crossbow [8]. Two of these motes are the MICA2 and the MICAz. Both have the same microprocessor in the 8-bit Atmel ATMega 128L which clocks in at 8MHz [3]. The current draw for operation is 8mA at full operation and less than 15uA in sleep mode. The next generation mote as of 2007 is the Imote2 which is also made by Crossbow. The Imote2 has the Intel PXA271 XScale 32-bit processor running at 13 to 416MHz. The current draw at full operation is 31mA and 290uA in deep sleep mode at 13MHz. Current draw goes up with the clock frequency.

The MICA2 has the CC1000 radio chip made by ChipCon [7]. The chip is capable of two different frequencies of 433 and 915 MHz with a link capacity of 38.4kbps using Manchester encoding. The MICAz and Imote2, on the other hand, have the CC2420, which is also made by ChipCon and is ZigBee compliant. It operates in the 2.4GHz band and has a link capacity of 250kbps using spread spectrum and O-QPSK modulation for tolerance to interference. The CC1000 has a maximum outdoor range of 300m with a current draw of 26.7mA (10dBm). The CC2420 has a maximum outdoor range of 125m with a maximum current draw of 17.4mA (0dBm). These ranges are based on the receive reception strength threshold. Reception power is based on the path loss model of $\frac{Pt}{d^\alpha}$ where α is a value equal or greater than 2 and Pt is the transmit power. Threshold power reception for the MICA2 and MICAz/Imote2 is -98dBm and -94dBm, respectively.

Additionally, EEPROM memory access also cost power. A read takes 565us at 6.3mA and a write takes 12.9 ms at 18.4mA. These values were taken from [13] for the MICA2 mote. With the information from above we can calculate the joule cost of transmitting one bit compared with the joule cost of an add instruction.

By looking at compression techniques we can calculate the potential power saved. This power savings potential is highly dependent on the the characteristics of the mote (processor/radio power draw), the data set, and the power characteristics and compression capability of the assistance hardware. These characteristics will tell whether compression should take place to save power.

It should be noted that sensor networks are generally delay tolerant. Many designs of protocols have traded off delay (as well as throughput) to save power. In the case of compression, better savings can be achieve with larger data sets (Section 3 goes into this in detail). Typical sensor readings should have a Gaussian distribution with low entropy in the overall data set. Thus, a sensor mote can make readings for several hours or days and send back aggregate compressed data. Not only does this save bytes by compressing data, but also the overhead endured by headers. In the case for the MICAz, headers from the physical (CC2420), MAC (tinyOS), and network layer (routing engine) can account for more than 23 bytes.

The novelty of this project is the inclusion of reprogrammable logic in a sensor network. To the best of

the author's knowledge this has not been attempted in the sensor network community due to the power hungry nature of standard FPGAs. As discussed in the sections to come the initial development is based on both FPGAs and CPLDs. A review of the current generation of hardware by Xilinx [15] and Altera [1] is given, followed by a power analysis of the consumption of different families of reconfigurable hardware. FPGAs have disfavorable characteristics because of their power draw and the need to be reflashed after being powered down. While it is unrealistic for FPGAs to be considered as a feasible components for a mote like devices due to its power draw, it does serve as a means of comparison to low power reconfigurable CPLD devices. Reprogrammable logic, such as Xilinx's CoolRunner II CPLD has a standby power draw as low as 17uA [15], and does not require expensive reprogramming when powered down because of the use of nonvolatile memory. Analysis of power is a two fold approach because it must look at what is the current cost of operation for a mote and the additional cost of added hardware. This paper analyzes the area and power constraints of CPLDs and looks to see if specialized hardware for data compression may provide better compression at lower power cost for a net power gain.

Section 2 will discuss the approach of analyzing power consumption and the cost of additional specialized hardware. Section 3 will talk about the applications for which compression is a beneficial and speaks of the data sets used by the compression algorithm presented in Section 4. Section 5 gives results on using the compression algorithm with the data set presented from Section 3. Section 6 has information about related work. Concluding remarks and future work is in Section 7.

2 Approach/Motivation

In this section the power values for the different processors and radio chips are given. As mentioned in Section 1, the radio chips are CC1000 and CC2420, which are two of the more popular radio chips. Moreover, the processors analyzed are the Atmel AVR ATmega128L 8-bit and the Intel XScale PXA271 32-bit processor. It should be noted, that the PXA271's datasheet could not be found, so much of the data presented here for that processor is based on estimates from the Crossbow website [8]. The data serves as motivation as to why it is useful to do compression, rather than just send raw data. Typically, compression has been done in order to save memory and hard disk space. The results shown in this section give the power taken in joules per instruction as well as the amount of energy taken for receiving and transmitting packets. This information gives hints on the viability of compression for low power devices.

2.1 Power overview

Just as a brief overview of the model of power consumption for the devices used, I review typical units of power here. If you are familiar with basic power terminology, you may skip this brief subsection.

A battery such as a double-A provides 1.5 volts and a current of 2000 miliamp hours. This means that the battery is capable of providing 3 watts of power for one hour. A joule is considered the amount of watts per second. A double-A battery can provide 10,800 joules over its lifetime. These numbers should be taken into mind when looking at the numbers presented in the next subsection.

2.2 Power Consumption

The values from Table 3 give the characteristics of both radios in terms of timing. The CC2420 has a much higher data rate and can thus get more bits transmitted at a cheaper cost. But because of the higher frequency the range is less. The values from Table 3 gives the power consumption per bit listed in Table 4. The power consumption of idle listening and receiving are the same, and the transmission varies depending on the power chosen for transmitting. The power of transmission can by dynamically changed on these motes, which serves well for topology control to save power and minimize interference to other nodes. Because the cost of idle listening can become very expensive in the lifetime of a mote, they must either be put into sleep mode, where the radio is turned off (after some sort of scheduling) or put into a polling phase, where the mote listens a small

Instruction	Cycles	Power
Add	1	3nJ
Mul	2	6nJ
Jump	3	9nJ
Store	2	6nJ
Load	2	6nJ
AND/OR	1	2nJ

Table 2: Power consumption of ATmega128L instructions and cycle count.

Radio Chip	CC1000	CC2420
Data Rate	38.4kbps	250kbps
Tx Time per Bit	26us	4us

Table 3: Operating modes of CC1000 and CC2420.

percentage of the time to overhear the preamble of a message. Generally, after a mote has heard messages over the air, the mote will go into full listening mode, and back to a polling mode shortly thereafter.

Table 1 shows the characteristics of the current generation processor as well as the next generation processor for sensor network motes. The newer generation provides 32 bit processing as well as a faster frequency. But it still comes at the expense of more current draw for both regular operation and sleep mode. To scale the two processors in terms of power per instruction, the ATmega128L spends 2.98nJ per cycle, whereas the PXA271 spends 10.73nJ, but the fact that the PXA271 is 32 bits should be considered in the comparison (4 times the operational bits for 3 times the power cost). Looking at Table 2, each add cost one cycle. Compared to the transmission of one bit from Table 4, a bit transmitted at full power for the CC1000 and CC2420 is 695 and 69 times more costly, respectively. A sensor network is generally built on multiple hops, and therefore even if the cost of doing more instructions gives a local loss in power, the overall system power still benefits.

3 WSN Application

Applications in WSNs are in many domains. These domains include environmental monitoring, animal tracking, and military surveillance. Many applicational uses are delay tolerant, which means data is not required in real time. Each autonomous mote can store sensed values and send aggregated data back to the base station as needed. Additionally, for real time usage, when the current sensed readings have deviated from the expected values, the collector may want a higher rate of sensing. In this case, so long as the hard deadline is no longer than the time it takes to collect a bulk set of data, compression can be applied. Most applications that are in use do not have hard deadlines, and this flexibility can be used to conserve power.

3.1 Data Sets

The mote application is for the MICAz platform. The mote runs an application executing in TinyOS [14] and is written in nesC [10]. TinyOS combines reusable components and "wires" them together. nesC is a language very similar to C but is event driven and lacks dynamic memory allocation.

In order to collect data sets which were used for testing the compression algorithm an application for sending data values was written for the MICAz using TinyOS. The MICAz has a stackable hardware architecture allowing for add-on sensor boards through a 51-pin connector (this is how compression hardware would be attached). The MTS300 sensor board was the means of sensing. It can measure light, temperature, and acoustics. In addition to writing a data collection application, a collecting base station application was used to transfer the collected data to the PC. The gateway is a MIB510 programming board which has a connected base station mote. The mote forwards traffic from the air over the UART and vice versa. The MIB510 transfer the packet to and from the UART and the serial port. A PC application logged data for light and temperature.

The mote application written collected 132 readings of two bytes each for a total of one flash page (264 bytes). The data was collect from a mote running an application which sensed light and temperature in an indoor office environment at a sampling rate of one second. Section 5 gives the results on the amount compressed. Different sampling rates and different

uProcessor	ATmega128L	PXA271
Operational Current Draw	8mA	31mA
Sleep Current Draw	15uA	390uA
Bits	8	32
Clock Freq	8MHz	13MHz
Cycle Time	124ns	76.92ns
Voltage	3V	4.5V

Table 1: Operating characteristics of the ATmega128L and PXA271 processor.

Radio Chip	CC1000		CC2420	
	Current	Energy/Bit	Current	Energy/Bit
Listening/Rx	74mA	88nJ/b	19.7mA	236.4pJ/b
Polling(1/100)	74uA	N/A	197uA	N/A
Tx @ -25 dBm	N/A	N/A	8.5mA	102nJ/b
Tx @ -20 dBm	5.3mA	414nJ/b	9.2mA	110.4nJ/b
Tx @ -15 dBm	7.4mA	578nJ/b	9.9mA	118.8nJ/b
Tx @-10 dBm	7.9mA	617nJ/b	11.2mA	134.4nJ/b
Tx @ -5 dBm	8.9mA	695nJ/b	13.9mA	166.8nJ/b
Tx @ 0 dBm	10.4mA	812.4nJ/b	17.4mA	208.8nJ/b
Tx @ 5 dBm	14.8mA	1156nJ/b	N/A	N/A
Tx @ 10 dBm	26.7mA	2086nJ/b	N/A	N/A

Table 4: Operating modes for transmission and reception of CC1000 and CC2420. Fields which are N/A are not applicable for those radios.

types and sizes of data is the subject of future work.

3.2 Transformed Data

In order to get good compression ratios, tricks can be played on the data to give it favorable characteristics for compression. For the case of sensor data, it can be noted that data is relatively similar from one sample to the next. Being so means the most significant digit will be the same over many samples. Data can thus be transformed as shown in Figure 2. The figure shows the most significant bytes reoccurring for each reading (every row). Swapping certain positions in the data creates patterns, allowing for greater compression.

4 Compression Algorithm

This section discusses the design, implementation, and testing of the compression hardware which was fitted on different reconfigurable platforms (See Section 5 for the types of hardware).

4.1 Design

The choice of the compression algorithm is based on the key metric of energy. When choosing the algorithm the author did not take into account what would transfer well over to hardware, but rather what would fit the application domain of sensor network data. Moreover only lossless compression algorithms were considered. These may be limitations and other types of low power compression is a subject of future work.

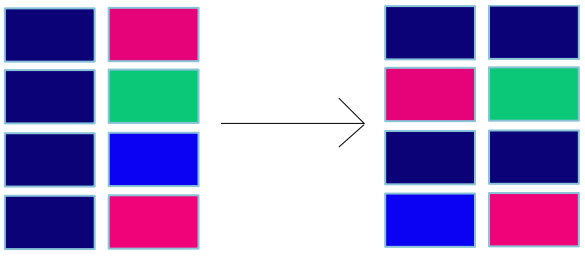

Figure 2: Transformation of the data set can result to additional patterns, allowing for better compression.

Much of the previous work on compression is focused on memory savings rather than energy. As mentioned in Section 2, the key goal is to minimize the amount of sent and received data, while maintaining a low overall cost of compression. The cost of compression comes from its computation expense on the mote, whereas the cost of decompression does affect the design decision because it is done on high powered nodes (sink nodes). Communication overhead is also a factor. In compression schemes such as Huffman encoding, the dictionary used must be known ahead of time. The sharing of the dictionary is additional overhead, and it also limits the adaptability of the algorithm to different data sets. Additionally, the design of the algorithm must be robust to packet loss. Sensor networks communicate in a medium where packet loss can be high and reception can not be guaranteed.

Thus to recap, the algorithm must be energy efficient, lossless, computationally low, low or no communication overhead, adaptable to different data sets as well adapt to changes for long term sensing. Lastly, the algorithm must be robust to packet loss.

The algorithm chosen is based on LZW compression. This compression builds a dictionary as data comes in serially. An eight bit word comes in and a 9 bit word is encoded, where many eight bit words can be encapsulated in the 9 bit word. Table 5 an example of input data and the corresponding encoded output. A flow chart can be found in [12].

4.2 Implementation

This subsection describes the implementation of the hardware module. The module was written in Verilog.

Figure 3 shows the top level design of the compression algorithm. The input to the circuit is an 8 bit word. The output is 9 bits and is latched when the ready signal is high. A done signal signifies when all the input stream has been encoded. The dictionary resets every 264 bytes to allow for robustness to packet loss between sets and adaptability for changing data.

Input Stream	Output	New Dictionary Entry
a		
aa	0	aa as 256
aa		
aaa	256	aaa as 257
ab	0	ab as 258
ba	1	ba 259
aa		
aac	256	aac as 260
	2	

Table 5: Example of LZW compression. Input Stream: aaaabaac, where a=0, b=1, c=2.

The implementation was done originally for the CoolRunnerII which has 256 macro blocks, where a macro block is approximately 1.3 logic elements. The size of the compression algorithm proved to be too large for this chip. This also eliminated the design to include interface logic. The supplied I2C interface for the chip by Xilinx takes up half of the logic for the CoolRunnerII. The focus of the implementation was switched to the MAX II and the Cyclone II by Altera (2210 and 33216 logic elements, respectively). The author used both the ISE 9.1i tool by Xilinx and Quartus II by Altera for synthesis. This allowed the author to notice the vendor specific Verilog constructs. The Verilog was written to synthesize in both environments for portability. The code is available at www.cs.ucsb.edu/~nchohan/comp.

4.3 Testing

Quartus II was initially used for debugging and testing. Creating test benches are easy with the graphical tool, yet analyzing the results is entirely wave based and slows development down. ModelSim was used instead to take advantage of the capability to monitor signals via print statements. This allowed for quicker development time. Xilinx's tool lacked a means of simulating, forcing the author to rely on third party tools. Test benches were created from the data sets via a perl script.

5 Results

Results are presented for two reconfigurable hardware platforms, the Cyclone II FPGA, and the MAX II CPLD. These platforms are from Altera [1]. Power estimates for the two boards were attained from the Quartus II PowerPlay feature.

As mentioned in Section 3 two data sets were attained. One for light and another for temperature. Each one of these sets also had a transformed version, using the tactic shown in Figure 2.

Due to the limited amount of logic in CPLDs (maxing out at 2210 logic elements) the number of entries which can be stored is limited. The MAX II EPM2210G CPLD had a maximum of 44 entries. The Cyclone II was able to fit the maximum amount, which can be attained with 9 bit output, of 256 new entries. Figure 7 shows the results for the MAX II CPLD. This can be compared to the results for the Cyclone II in Figure 8. Because of the lack of entries for the CPLD, the compression actually looks like decompression for non-transformed temperature data. The FPGA is able to achieve compression for all data sets. An oddity is the fact that the transform data actually is worse for the light data set, whereas it works well for the temperature data set. The FPGA's success shows that CPLDs are in need for more logical units. A comparison of the clock rate, power consumption and logical units is shown in Figure 6. The clock rate is low for the FPGA and uses more power. Disregarding the number of logic elements we can see

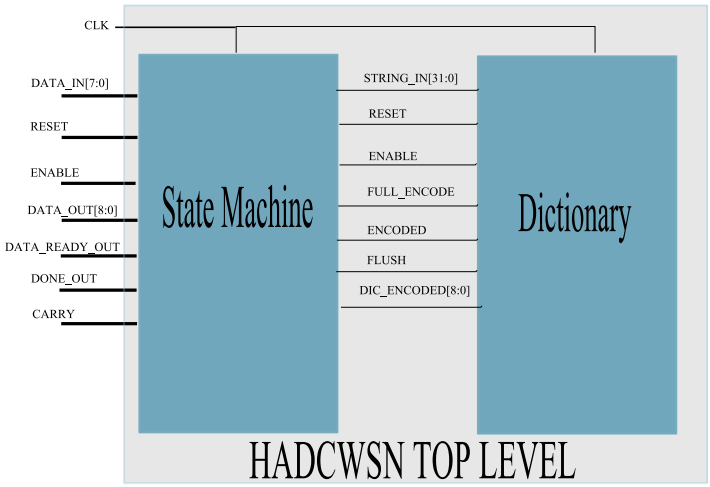

Figure 3: Top level design of hardware compression module.

Platform	Power	Rate	Logic Units	Entries
Cyclone II	112mW	12.88MHz	11973	256
MAX II	10.29mW	36.11MHz	2143	44

Table 6: MAX II EPM2210G CPLD and Cyclone II design specifics.

Data Set	Input Bytes	Output Bytes
Temperature Normal	264	279
Temperature Transformed	264	221
Photo Normal	264	129
Photo Transformed	264	140

Table 7: MAX II EPM2210G CPLD compression capability with 44 dictionary entries.

Data Set	Input Bytes	Output Bytes
Temperature Normal	264	237
Temperature Transformed	264	212
Photo Normal	264	121
Photo Transformed	264	132

Table 8: Cyclone II FPGA compression capability with 256 dictionary entries.

that the Cyclone II would take 2.35 microjoule and the MAX II would take 964 nanojoules. These values are functions of the clock rate, power consumption, and the number of cycles to compress one flash page (264 bytes or approx. 270 cycles). This comparison is not fully fair due to the additional entries the Cyclone II design has. Nonetheless, these values should be compared to the number presented in Section 2 to see if compression is advantageous.

The numbers from Table 3 shows that each bit cost 812.4 nJ and 208.8 nJ for transmitting at 0dBm, for the CC1000 and CC2420 respectively. As mentioned before, the cost of compression for the CPLDs operation is approximately 964 nJ. The average number of bytes saved across data sets was 79 bytes or 632 bits. Looking at power from a transmit perspective shows that it cost less than 1 bytes worth of power (6.50 uJ for CC1000 or 1.67 uJ for CC2420) in order to reduce the total number of bytes by 79 bytes. This gives a net savings of 78 bytes or a 29.5 percent savings of total transmission power. Because transmission power is the bulk of power draw, this 29.5 percent can be a major portion of total battery consumption and can give significant improvement in total battery life. The average compression is greater by 9 bytes using the bigger design in the FPGA and the cost of compression is 2 times greater. Additionally, the FPGA has a high standby cost, or if completely powered down, would need to be reflashed.

The numbers given in this section do not account for the power consumed by the interface cost. Moreover, the numbers assume that the hardware is being clocked at the maximum given frequency as specified by the Quartus toolset, which is actually much higher than the frequency the processor runs at (8MHz versus 36.11 MHz). All presented values are estimates from simulation and CAD tools and actual empirical measurements are needed to verify these results.

6 Related Work

Previous work for compression in sensor networks has been primarily done using the microprocessor on the mote and to the best of my knowledge has not been offloaded using additional hardware. Much work has gone into devising lossless energy aware compression algorithms such as work by [5]. They looked at several different compression algorithms and the power consumption of each on variable data sets. Much like this paper it looks at the substitution of execution of instructions for the reduction in the number of bytes transmitted. Their results showed that there are many variables to having successful power aware compression. Things depend heavily on the relative energy cost of the CPU, memory, network, data and the compression ratio achieved. In an ad-hoc 802.11 network it may be the case where an individual node may have to spend more energy to transmit so that the overall system saves power. Their results were not for a sensor network, nor did they use specialized hardware. A paper for compression in sensor networks was done by [12] for data compression in delay tolerant networks. They address the same problem of power constraints through compression as this paper does. They see that energy benefits grow as the number of hops does. Moreover, retransmissions must occur because the communication between hops is lossy. Shorter packets are less likely to be corrupted because they are in the medium for a shorter amount of time, and less power is exerted when a retransmission does occur. They differ from [5] in the fact they propose techniques and algorithms tailored for sensor nodes which are power and memory constrained. They focus on LZW compression which is based on dictionary entries which are used for lookups. There is no cost of synchronizing dictionaries between the encoder and the decoder because they build the dictionary as data comes in. The main constraint is the fact that the dictionary takes up significant RAM. They use a 256 entry dictionary as done in this paper. Additionally, they have a mini-cache to take advantage of the fact that sensor readings are spatially correlated over short periods of time. They suggest aggressive compression, even at the cost of a loss of power at the local node, in order to save power downstream and for the overall system.

Other types of compression in sensor networks have been based on the spatial and temporal redundancies of sensor readings. Tactics such as funneling, where nodes send their data to a parent which compresses and does in network processing of the data, reduces the overall transmission cost to the sink node [9].

Pipeline in-network compression as presented in [2] is a delay tolerant method which has each node compress duplicate readings of sense values, and it minimizes the total number of packets to reduce the overhead of packet headers. The compression presented is very lightweight, and depends heavily on values of the sense data to be similar across time stamps. Other schemes for data compression in a WSN include distributed compression as presented in [4, 11].

The hardware assisted low power compression has been presented by Benini et al in [6]. The adaptive compression is based on statistical analysis of the data. The data dictionary, for their static approach is built using offline data profiling, which is something which is taken into consideration in devising this paper's compression techniques. Generally, sense data is expected to have a Gaussian distribution, which can be exploited for savings in data transmission. Benini et al shows energy savings of 39% for the offline approach, and 27% for the adaptive approach for their set of benchmarks. This paper show similar savings using an adaptive approach of 29.5%.

7 Future Work

Future work entails implementing other low power compression algorithms as well as analyzing different and more realistic data sets. Because more technology is coming out for low power FPGAs, a new area of reconfigurable hardware over the air may become a viable option. From the experience gained from this project the author may try to take it a step further and attempt to build dynamic and reconfigurable hardware in sensor networks but with the current limitations of both CPLDs and sensor motes, this is very challenging. The major problem is power consumption, so it will depend on how expensive it is to reconfigure the hardware over the air, as well as what does this extra flexibility buy us in the sensor network application domain.

References

[1] Altera Corporation. `http://www.altera.com`.

[2] T. Arici, B. Gedik, Y. Altunbasak, and L. Liu. Pinco: A pipelined in-network compression scheme for data collection in wireless sensor networks, 2003.

[3] Atmel corporation. `http://www.atmel.com`.

[4] Waheed Bajwa, Jarvis Haupt, Akbar Sayeed, and Robert Nowak. Compressive wireless sensing. In *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, pages 134–142, New York, NY, USA, 2006. ACM Press.

[5] Kenneth Barr and Krste Asanovi. Energy aware lossless data compression. In *MobiSys '03: Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 231–244, New York, NY, USA, 2003. ACM Press.

[6] L. Benini, D. Bruni, A. Macii, and E. Macii. Hardware-assisted data compression for energy minimization in systems with embedded processors. In *DATE '02: Proceedings of the conference on Design, automation and test in Europe*, page 449, Washington, DC, USA, 2002. IEEE Computer Society.

[7] Chipcon corporation. `http://www.chipcon.com`.

[8] Crossbow Technology. `http://www.xbow.com`.

[9] K. Ramchandran D. Petrovic, R. C. Shah and J.Rabaey. Data funneling: Routing with aggregation and compression for wireless sensor networks. In *First IEEE International Workshop on Sensor Network Protocols and Applications*, May 2003.

[10] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesc language: A holistic approach to networked embedded systems. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 1–11, 2003.

[11] J. Kusuma, L. Doherty, and K. Ramchandran. DISTRIBUTED COMPRESSION FOR SENSOR NETWORKS. pages 82–85.

[12] Christopher M. Sadler and Margaret Martonosi. Data compression algorithms for energy-constrained devices in delay tolerant networks. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 265–278, New York, NY, USA, 2006. ACM Press.

[13] Victor Shnayder, Mark Hempstead, Bor rong Chen, Geoff Werner Allen, and Matt Welsh. Simulating the power consumption of large-scale sensor network applications. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 188–200, New York, NY, USA, 2004. ACM Press.

[14] TinyOS. `http://www.tinyos.net`.

[15] Xilinx corporation. `http://www.xilinx.com`.