Adaptive Lossless Compression in Wireless Body Sensor Networks

Saad Arrabi, John Lach
Charles L. Brown Department of Electrical and Computer Engineering
University of Virginia, Charlottesville, VA 22904 USA
+1-434-924-6086
{arrabi, jlach}@virginia.edu

ABSTRACT

In most wireless body sensor network (BSN) applications, the vast majority of the total energy is consumed by the wireless transmission of sensed data. Transmitting one bit using a typical wireless communication system can consume as much energy as 1000 cycles of an embedded processor. Reducing this transmission energy – even at the expense of increasing another component's energy – is essential to meeting the battery life and form factor (i.e. small battery) requirements of many BSN applications. While improved wireless communication and networking techniques can help do just that, simply compressing the sensed data to reduce the number of transmitted bits can provide significant savings. However, BSN platforms and applications impose many constraints on compression techniques, including fidelity (focus on lossless techniques, as required for many medical BSN applications), programmability (enable ease of code development and deployment), adaptability (achieve high compression ratio regardless of location, subject, activity, etc.), and implementability (require low processing and memory resources). This paper analyzes variations of two known real-time lossless compression algorithms, Huffman encoding and delta encoding, within the context of these BSN constraints. Experimental results on a multi-node accelerometer-based BSN show the strengths and weaknesses of each algorithm and ultimately reveal the superiority of dynamic delta encoding for BSNs, including an average 35% energy savings across a range of activities, sensor locations, and sensor axes.

1. INTRODUCTION

Wireless BSNs are emerging as a technology with tremendous potential for a variety of applications, including healthcare, clinical medicine (including telemedicine), biomedical research, emergency medicine, first responder safety, homeland security, athletics, etc. While significant efforts have been made to develop and deploy BSNs, there are numerous technical challenges that remain in order for BSNs to be practical for the applications for which they are envisioned. In particular, BSN nodes must be smaller to minimize invasiveness and maximize wearability and must have longer battery lifetimes to enable significant data collection. However, these are competing metrics, as the size (and therefore capacity) of the battery is the primary factor in determining the dimensions of a BSN node. While improved energy harvesting and storage provide promise for the future [9], the options immediately available to BSN developers involve improving energy efficiency – especially the energy related to the wireless transmission of sensed data, which is the largest energy consumer in most BSN systems.

While work is being done to improve the energy efficiency of wireless transceivers and to explore the energy vs. quality-of-service tradeoffs in communication coding and wireless networking protocols, the most direct way to reduce energy due to wireless transmission is to simply reduce the number of bits that need to be transmitted. On-node signal processing algorithms, such as feature detection and pattern recognition, can be used to convert some of the raw sensed data into application information that can be coded using fewer bits, but most of these algorithms are application-specific. It is desirable from a programmability and deployment perspective for BSN devices to be general enough that a single device executing the same code can be efficient for a wide range of applications and even sensor locations and activities within applications. Even within a particular application, the critical data/information cannot always be reliably determined/extracted, so any such pre-processing may be problematic for fidelity-critical applications, such as the many medical applications for which BSNs are envisioned. Therefore, lossless compression techniques are the most general and reliable tools for reducing wireless transmission energy in BSNs.

However, given the severe resource constraints of most BSN nodes, any compression algorithm must have low processing and memory requirements while still providing real-time throughput (i.e. the processing must keep up with the data sampling rate) and an overall energy savings (i.e. the additional energy consumed by the processor running the compression algorithm cannot exceed the wireless transmission energy saved). The embedded processors on BSN nodes typically operate in the tens of megahertz and have on-chip memories of only several kilobytes. These restrictions can limit the implementability of many dictionary-based compression techniques (e.g DEFLATE, RLE). Efforts to increase total memory by including off-chip memories have been proposed [12], but off-chip access time can be quite high and the additional resources (particularly area and power) may not make this approach worthwhile, especially in resource scarce BSNs. Finally, the characteristics and requirements of many BSN applications are significantly different from most WSN applications, including both static parameters (e.g. higher

BodyNets'09, April 1–3, 2009, Los Angeles, CA, USA.

Digital Object Identifier: 10.4108/ICST.BODYNETS2009.6017
http://dx.doi.org/10.4108/ICST.BODYNETS2009.6017

data rates and fidelity requirements) and dynamic variables (e.g. rapidly varying data properties and channel conditions).

This paper provides an analysis of resource-aware lossless compression techniques specifically targeting BSN characteristics and requirements. The following metrics are considered in the analysis:

- Compression ratio: Number of bits before compression divided by the number of bits after compression.
- Processor cycles: Number of cycles the processor needs to run the compression algorithm for each sample, which is proportional to the processing energy consumption of the algorithm.
- Average energy savings: Combination of the previous two metrics that is estimated from the data sheets of common BSN transceivers and embedded processors.
- Memory requirement: Approximate amount of memory required to implement the algorithm.
- Adaptability: Ability of the compression algorithm to adapt to static parameters (application, wearer, sensor type, sensor location) and dynamic variables (activities, time), continuing to provide high performance without negatively affecting the other metrics of interest.
- Programmability/deployment: Related to the previous metric, the ability to adapt to static parameters enables the developer to program and deploy all nodes the same way, regardless of the application, wearer, sensor type, and sensor location.

Background work considered a number of lossless compression techniques, but this paper focuses on two of the most promising with respect to the metrics of interest – delta encoding and Huffman encoding. Delta encoding transmits the difference between each reading rather than the full reading. If the number of bits required to encode the delta (referred to as delta bits) is regularly less than what is required to encode the full reading, significant compression ratios can be achieved. The compression can be made lossless by including a special code after a reading that exceeds the representable delta range. The number of delta bits can be determined statically or dynamically, and both approaches are considered here. Huffman encoding depends on some readings occurring more than others, so by assigning frequent readings codes with fewer bits, the total number of transmitted bits will decrease. This paper compares both algorithms and some of their derivatives within the context of BSNs and with respect to the above metrics.

The rest of this paper is organized as follows. Section 2 describes some related work done in the field. Section 3 provides additional detail on the BSN domain specific metrics by which the compression techniques are evaluated, and Section 4 presents some background information about the compared compression techniques. Section 5 specifies the experimental setup, including the accelerometer-based BSN that provided the sample experimental platform for this study, and Section 6 details the experimental results and provides analysis on the compression techniques with respect to the metrics.

2. Related Work

While many previous results evaluate compression techniques, few evaluate them from sensors network perspective. For BSNs, the focus must be on energy and other resource requirements rather than solely the compression ratio.

Some compression techniques have been modified to suit general sensor network applications. One group focused on exploiting the spatial correlation in sensor data [4][6], but those techniques are directed towards specific applications. Other researchers tried to exploit the spatio-temporal correlation by focusing on data-centric routing and aggregation [1][17].

Some researchers tried to modify existing techniques to suit sensor networks, such as Saddler and Martonosi's work focusing on LZW compression [19] and some of its derivatives aimed at embedded systems [4]. In their research, they show that such techniques, while suitable for embedded systems, are still not suited for sensor systems in general [16]. They looked into several compression techniques like LZO, ZLib, bzip2 and PPMd, and they concluded those techniques will not be suitable for sensor networks because they need more than 10 kB of RAM, which is the typical RAM size in many sensor nodes. As a result, they proposed a technique derived from LZW, called S-LZW.

In this paper, we selected off the shelf techniques that showed potential to be suitable for BSNs. In the same time the techniques are general enough to be application independent.

3. METRICS

Normally to evaluate a compression technique, only the compression ratio is taken into consideration. However, like in WSNs, compression in BSN devices must consider several other metrics [15]. In this section, we detail a mix of metrics that provides an overall performance evaluation for compression techniques within the context of BSNs.

3.1 Energy Savings

Since the main point of implementing a compression algorithm directly on BSN sensor nodes is to reduce energy consumption, a formal energy savings equation must be introduced. Given that the embedded microprocessor on the sensor node may be in a sleep mode if it were not being used for compression, the energy equation must include both the reduction in transmission energy due to the reduced number of transmitted bits and the increase in processing energy. Given that compression does not affect other sources of energy consumption (e.g. the energy drawn from the sensors), only those two sources are considered here.

The average energy savings per sample is simply the difference between the energy before and after compression:

$$E_{Before} = RES * E_{Bit} + CYC * E_{Idle} \quad (1)$$

$$E_{After} = \frac{RES}{CR} * E_{Bit} + CYC * E_{Active} \quad (2)$$

where E_{Before} and E_{After} are the energy consumption per sample per sensor before and after the compression, respectively, RES is the number of bits for each reading (i.e. bits per sample per sensor), CR is the compression ratio, E_{Bit} is the energy required to transmit one bit wirelessly, CYC is the number of active processor cycles the compression algorithm needs to compress one reading, and E_{Idle} and E_{Active} are the energy per idle and active processor cycle, respectively. RES, E_{Bit}, E_{Idle}, and E_{Active} are specific to the BSN platform – specifically the transceiver and the embedded microprocessor. CYC is a function of both the compression algorithm and the microprocessor, and CR is as function of the compression algorithm and the actual data. As discussed in Section 6, this paper uses a custom accelerometer-based BSN

Digital Object Identifier: 10.4108/ICST.BODYNETS2009.6017
http://dx.doi.org/10.4108/ICST.BODYNETS2009.6017

platform in a motion capture application to obtain all of these values.

Even within a given BSN platform, E_{Bit} may vary based on a number of factors. Wireless devices typically stay connected all the time, consuming energy constantly, and the energy consumption surges during transmission. In Equations 1 and 2, E_{Bit} represents the difference between the "stay connected" energy and the "transmit" energy. In order to break down this energy on a per bit basis, one must also consider the transmit rate and packet size. This paper assumes a transmit rate of 115.2kbps and the maximum packet length, which minimizes the per bit transmission energy. This maximum packet length will be maintained regardless of the compression ratio, so E_{Bit} is reduced due to the lower packet rate.

Many embedded microprocessors are programmed to go into a sleep mode (i.e. a very low power mode) when no processing is required, so the additional active cycles required – and processing energy consumed – by the compression algorithms are considered in the energy savings, as $E_{Active} >> E_{Idle}$. However in many systems, the overhead of going to sleep and waking up does not justify the often short amount of time spent in the sleep mode. In such cases, the processor will perform NOPs until functional processing is again required. NOPs often consume almost as much energy as a functional cycle, so E_{Idle} almost equals E_{Active}. In such systems, the additional processing energy consumed by the compression algorithm is negligible.

In most BSN systems, $E_{Bit} >> E_{Active}$, often by several orders of magnitude. However, if the processing load imposed by a compression algorithm is significant, the additional processing energy can be significant. For example, Algorithm A may provide a higher compression ratio than Algorithm B, but if the complexity of Algorithm A is significantly higher, Algorithm B may actually provide greater total energy savings. An example of this is demonstrated in Section 6.2.

3.2 Resource Requirements

Since any compression algorithm will be implemented on the embedded processor, the algorithm will be constrained by the processor's limited resources. The processors used on BSN platforms are typically significantly smaller and less capable than those used in WSNs, making algorithm implementability a significant constraint.

The processor's memory imposes one of the key constraints, as many embedded processors that are appropriate for BSNs have only a few kilobytes of memory. Algorithms that use tables, trees, or dictionaries could easily exceed this memory restriction and therefore be excluded from use in BSNs.

Algorithms must also operate under the limited throughput capabilities of the processor, which typically has a relatively simple datapath and runs at tens of megahertz. Depending on the sampling rate of the BSN platform (the accelerometer-based node used in this study samples three sensor axes at 120 Hz, resulting in 360 readings per second), it may be difficult for a compression algorithm to be executed in real-time, which is a hard requirement.

3.3 Adaptability

It is highly desirable to have a compression algorithm for a BSN platform that will perform well for any application, wearer, sensor location, activity, etc. While different compression algorithms can be programmed onto each BSN node based on these factors, this requires significant additional programming and deployment effort; not to mention the challenge of profiling that would be required to determine the appropriate compression algorithm for each scenario. In addition, the data being collected by a BSN is rarely static, as the wearer is often performing different activities that change the compression capabilities of the implemented algorithms.

Instead of using such static techniques, BSN compression algorithms should have the ability to adapt and perform well across different

- applications,
- test subjects (wearers),
- sensor locations and orientations,
- axes of the same sensor, and
- activities over time.

As shown in Section 6, the performance of compression algorithms across these static and dynamic variables can vary greatly. While traditional WSNs also suffer from this problem due to node location and dynamic data, the effects are typically more extreme in BSNs. The overall energy efficiency of a BSN therefore depends on identifying the algorithm that performs the best on average across an entire data collection.

4. TECHNIQUES BACKGROUND

The work detailed in this paper included the consideration of a number of lossless compression algorithms. The two families that were identified as the most promising given the BSN metrics detailed above are Huffman encoding and delta encoding. Several variations of each are evaluated in Section 6.

4.1 Huffman Encoding

Huffman encoding leverages the uneven distribution of readings in datasets, using fewer bits to encode more common readings to achieve an overall compression. An imbalanced tree structure is generated based on the presumed frequencies of each reading, with high frequency readings at shallower leaf nodes than those occurring less often. Each reading's code is determined by traversing the tree from root to leaf, with each branch node providing one bit to the code. The length of each code is therefore determined by the depth of its leaf.

A number of practical issues make the use of Huffman encoding a challenge for BSNs. First, Section 6 reveals that many of the reading frequency distributions (using the accelerometer-based BSN platform and multiple sensor locations and activities) are relatively flat, thus limiting the compression capabilities of Huffman encoding. Second, while the computational complexity of the Huffman algorithm as it is running with an existing tree is small, the tree itself can be quite large and potentially exceed the memory constraints of many BSN embedded processors. This is especially problematic in lossless compression when every possible reading must be encoded and must therefore have a leaf in the tree, even if that reading is extremely rare. Finally, traditional static Huffman encoding depends on the existence of a

Digital Object Identifier: 10.4108/ICST.BODYNETS2009.6017
http://dx.doi.org/10.4108/ICST.BODYNETS2009.6017

reading frequency distribution to generate the tree, and the compression performance achieved depends on how well the dynamic data conforms to that frequency distribution. It is therefore essential that a BSN developer wanting to use Huffman encoding perform extensive data collections to profile the reading frequency distribution. As discussed in Section 3.3, this can be extremely difficult given the numerous static and dynamic variables. Therefore, this static technique's ability to perform well across many applications, wearers, sensor locations, sensor axes, and activities is limited.

It is therefore desirable to also consider an adaptive Huffman encoding technique that dynamically generates and alters its tree based on the actual reading frequency distribution as it occurs and changes. This does not require a previously constructed tree or any reading profiling. In the tree, each reading will carry its value along with the frequency of its use. This way, the tree can update itself, keeping the most frequent readings on the shallower levels to minimize their code lengths [8]. This adaptability can potentially provide higher compression performance across all of the static and dynamic variables without additional programming and deployment effort. However, as discussed in Section 6, the performance of adaptive Huffman encoding in a sample BSN application is limited due to a number of factors.

While this adaptive technique is significantly more computationally complex than static Huffman encoding, it can potentially be implemented in real-time on BSN embedded processors, although the number of processor cycles (*CYC* in Equation 2) required per reading may be relatively high. While adaptive Huffman encoding includes the storage of reading frequencies in addition to the coding tree, its total memory requirements are often smaller than those of the static technique. The static tree remains a fixed size in memory throughout execution regardless of the occurrence (or lack thereof) of certain readings. The adaptive technique can choose to discard certain readings that have not occurred for some time, keeping the tree size to some maximum memory requirement and reinserting a discarded reading should it reoccur in the future [16].

4.2 Delta Encoding

Delta encoding achieves compression by sending the difference between a reading and its predecessor rather than sending the full reading. This technique has been used effectively for a variety of applications, from images to web pages [10]. The compression rate is determined by the difference between the number of bits designated for conveying the difference (delta bits) and the number of bits required for the full reading. This technique is often used in lossy compression, as differences may occur that exceed the range that can represented by the number of delta bits. However, delta encoding can be made lossless by including a special overflow code in place of the difference, followed by the full reading. This lossless algorithm has been employed here for BSNs.

One of the challenges for both the lossy and lossless versions is the selection of the number of delta bits to be used. If too few bits are used, the lossy delta encoding will become extremely lossy (too many readings will be beyond the encoding range) and the lossless encoding may actually have a compression ratio that is less than one (many readings will both be transmitted in full and include the special overflow code). If too many are used, the compression ratio will be lower than what is possible. Like as is required to determine frequency distributions for the static Huffman tree, the BSN application must be profiled, and the collected data must be analyzed to determine the optimal number of delta bits. This again adds significantly to programming and deployment effort and is still limited by dynamic variables, such as different activities over time.

Delta encoding can also be made capable of adapting to static and dynamic variables, much in the same way as adaptive Huffman encoding but with significantly lower complexity. For a given interval of time (or number of samples), dynamic delta encoding determines whether it would have been better to use a different number of delta bits, and it sets the number of delta bits for the next interval accordingly, including a special code to indicate to the receiver that the number of delta bits has been changed. Equation 3 calculates the number of bits needed for encoding an interval of samples:

$$Bits = I * DB + OF * FR \tag{3}$$

where I is the number of samples in each interval, DB is the candidate number of delta bits, OF is the number of samples that results in overflow given DB, and FR is the number of bits in a full reading. Using this equation, the processor reevaluates DB every interval, always adjusting to maximize the compression ratio.

The computational and memory requirements of this technique are slightly higher than static delta encoding but are significantly lower than both static and adaptive Huffman encoding. The complexity depends on the range considered for delta bit alteration. As discussed in Section 6, this study determined that ±1 delta was sufficient to provide high performance.

5. EXPERIMENTAL SETUP

To compare between the identified compression techniques, we used the TEMPO BSN, shown in Figure 1, that measures linear acceleration in three axes [14]. The sampling frequency is 120 Hz, and the resolution of each sample is 12 bits per channel. Without any on-node compression, each node sends its 4320 bits per second over a Bluetooth wireless channel. The node has approximately 10kB of RAM for data. The compression algorithms were implemented on the resident MSP430F1611 embedded microprocessor. (The code is available online at http://www.ece.virginia.edu/inertia/embedded.php)

Figure 1. TEMPO BSN node

TEMPO nodes were attached at multiple points on the body (including the wrists, ankles, hip, and forehead) of a healthy 22 year old male. The results in Section 6 are for both a 45 minute

Digital Object Identifier: 10.4108/ICST.BODYNETS2009.6017
http://dx.doi.org/10.4108/ICST.BODYNETS2009.6017

recording of various movements and activities and shorter recordings of specific activities. The compression techniques described in Section 4 were implemented and evaluated with respect to the metrics described in Section 3. To measure the number of cycles each algorithm required, an implementation of the algorithm was mapped to the MSP430F1611. Then by switching an output pin at the beginning and at the end of the algorithm, we were able to calculate the number of cycles.

Many factors can affect the outcome of the compression, such as the processor, wireless device, packet size, transmission rate and memory. The results obtained in this paper are specific for this platform and the type of the data taken. The trend, however, should be similar on other platforms and other types of data.

6. RESULTS

Figure 2 shows the average compression ratios for each axis of selected sensor locations across the entire 45 minute dataset for adaptive Huffman, static Huffman, and dynamic delta encoding. The tree for the static Huffman was generated based on the probability distribution of the readings over the entire 45 minute dataset. Figure 3 shows the compression ratios for selected sensor locations for a 15 second window of healthy symmetric gait. The static Huffman tree was generated from the probability distribution over this specific 15 second window.

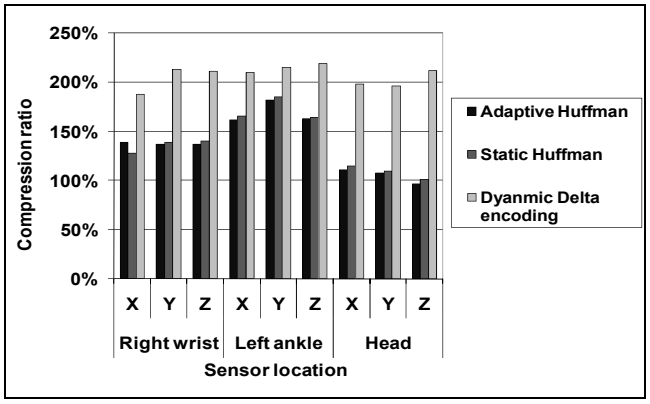

Figure 2. Compression ratios for entire 45 minute recording

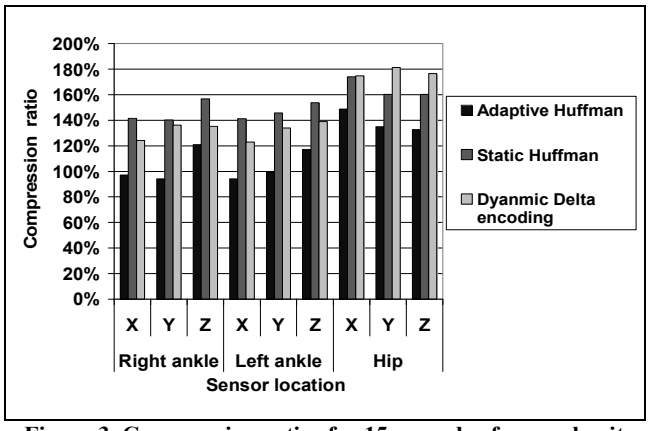

Figure 3. Compression ratios for 15 seconds of normal gait

6.1 Huffman Encoding

6.1.1 Static Huffman

As shown in Figure 2, across the entire collected dataset, sensor locations, and axes, static Huffman encoding was able to achieve an average compression ratio of approximately 135%. Even though the Huffman tree was constructed based on the actual dataset, it was not tailored to each of the individual activities and movements over the 45 minute period. Therefore, the frequency distributions of readings may have matched well during certain periods with the probability distribution used to generate the static tree (as was the case in Figure 3, when an average compression ratio of almost 160% is achieved), but that is not always the case, and the overall compression ratio suffers from this lack of adaptability. As expected, it was difficult to identify a single static probability distribution that was appropriate across a wide range of activities, axes, and locations, so the overall frequency distribution was used. It is possible to generate a number of Huffman trees from profiled probability distributions and invoke them at the appropriate time and for the appropriate sensor location and axis, but this is problematic with respect to programmability and deployment.

It is interesting to note in Figure 3 that Huffman encoding did better than dynamic delta encoding for the ankle sensors but not for the hip sensor. The hip acceleration is much less than the ankles and therefore requires few delta bits for encoding.

Another major issue regarding static Huffman is that it requires a relatively large amount of memory. For lossless compression, each reading value must have its own dictionary index, so a 12 bit resolution system like TEMPO has 4096 indexes. However, some applications may assume that only a subset of the reading values actually occur and simply assign one more index for all other values. The data collected during this study revealed that less than 1024 of the possible readings occurred, but that still requires that a large number of leaves and the accompanying tree structure be stored in memory. Some techniques can be used to minimize the memory requirements for such structures, but this is still likely to be problematic for most BSN platforms. In addition, the complexity of the tree search is high, which increases the number of active processor cycles (*CYC* in Equation 2) and can cause the throughput constraints of the system to be violated.

6.1.2 Adaptive Huffman

As described in Section 4.1, the adaptive Huffman technique is designed to address the static technique's lack of adaptability with the goal of dynamically tailoring the Huffman tree to the current reading frequency distribution. However, the compression ratios provided by adaptive Huffman encoding were mixed based on 1) the amount of time it took for a tree to be adapted vs. the amount of time the new tree provided good performance, and 2) the distribution of the readings to be encoded, with more even distributions providing lower compression ratios.

Both of these factors come into play when examining the performance of adaptive Huffman in Figure 2. Given the adaptability of this technique and the various activities that were performed over the 45 minute data collection (and the resulting various reading frequency distributions), one might expect that adaptive Huffman would perform significantly better than the static version. However, it is clear that the activities and resulting

Digital Object Identifier: 10.4108/ICST.BODYNETS2009.6017
http://dx.doi.org/10.4108/ICST.BODYNETS2009.6017

distributions were not held constant long enough to enable the Huffman tree to adapt to them and provide an extended compression ratio benefit. This is revealed in Figure 3, as the 15 seconds of walking does not provide enough time for the adaptive technique to settle on the appropriate tree and ultimately benefit from it. In fact, the compression ratio is < 1 for some sensor axes. It is likely that longer-term activities would result in better adaptive Huffman performance. In addition, some of the activities had relatively even reading frequency distributions on some axes and locations, resulting in lower compression rations regardless of adaptability. Finally, the memory and processing requirements of adaptive Huffman encoding also pose challenges, as described in Section 4.1.

6.2 Delta Encoding

It is clear from Figure 2 that dynamic delta encoding provides by far the best compression ratio for every sensor and axis over a long data collection that includes a variety of activities. Figure 4 shows how the number of delta bits changes for different activities over the 45 minute data collection (data from the right wrist), demonstrating the algorithm's ability to adapt to the current activity and achieve the highest compression ratio without having to change the algorithm. Figure 3 shows that it also does well over short periods of a single activity, even performing better than the optimized static Huffman for the hip sensor for the reason mentioned above.

In addition to providing the best compression ratio, it also requires little memory and the fewest processor cycles, further adding to its energy savings capabilities relative to static and dynamic Huffman. Finally, it uses the same simple algorithm regardless of the application, test subject (i.e. BSN wearer), sensor location, sensor axis, or activity, making it extremely flexible and adaptable and easing the programming and deployment. It is therefore the conclusion of this paper that delta encoding is the best algorithm for lossless compression in BSNs. The question therefore becomes one of optimizing dynamic delta encoding for the metrics detailed in Section 3.

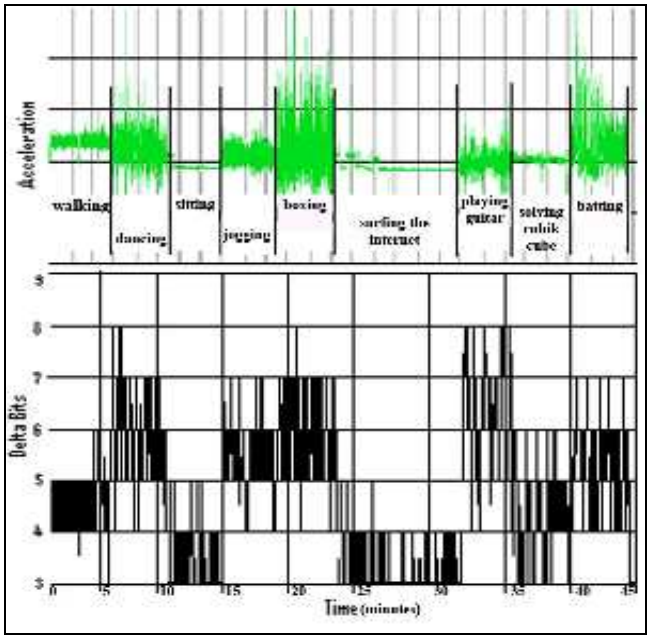

Figure 4. Number of delta bits across different activities

As mentioned in Section 4.2, the algorithm changes the number of delta bits every predetermined interval and does so by inserting a special code, which incurs an overhead. Therefore, changing the number of delta bits too often may reduce the compression ratio or cause rapid oscillations between settings. However, not changing often enough may reduce the algorithm's ability to adjust to rapid changes in the data characteristics. Figure 5 shows the compression ratio for the three axes on the right wrist over the 45 minute data collection period as a function of the delta bit update interval. It is clear that the optimal interval for all three axes lies between 0.25 and 1 second, and 0.5 seconds was selected for the rest of the results in this paper, including those in Figures 2 and 3. However, this optimal interval is data- (and therefore BSN system-, application-, wearer-, sensor-, and axis-) dependent, and the proper selection of the update interval is essential to compression performance.

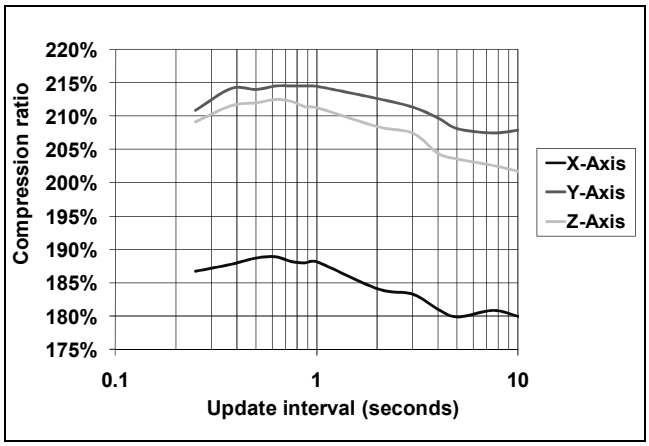

Figure 5. Compression ratio across different update intervals

Another interval-related issue is how large changes in optimal delta bits are handled. Consider, for example, the situation when the current number of delta bits is set to four and the data suddenly changes to include large deltas that require six or more delta bits for representation. Using Equation 3 and the ±1 delta bit options, it may be determined that the number of delta bits should be reduced by one rather than increased by one because the new deltas will overflow any of the available choices – three, four, or five delta bits. Given that every reading will result in overflow, the highest compression ratio will be provided by the fewest number of delta bits. Three approaches were considered to address this issue. First, the maximum change in the number of delta bits could be increased to ±2 or ±3, but that dramatically increases the computational complexity of the algorithm and is unlikely to provide a significant benefit for real BSN data streams. Second, the update interval could be increased in the hope that any dramatic change in delta sizes are not long lasting, but this cannot be guaranteed. Third, Equation 3 could be altered to only make changes in the number of delta bits when the benefits of the change exceed a defined threshold, but that does not guarantee that the algorithm will converge on the optimal number of delta bits over time. Combinations of these three methods were investigated, but none improved the compression ratios more than 2%.

Figure 6 compares the performance of dynamic delta encoding to two other variations of delta encoding. The first, passive delta encoding, always uses seven delta bits. Seven was selected based

Digital Object Identifier: 10.4108/ICST.BODYNETS2009.6017
http://dx.doi.org/10.4108/ICST.BODYNETS2009.6017

on an analysis of the deltas for all of the sensor locations and axes from several datasets. The second, optimized delta encoding, also uses a constant number of delta bits, but that number was determined for each individual sensor location and axis based on the deltas in the 45 minute dataset. Neither technique benefits from dynamic adaptation and, therefore, neither achieves the compression ratios provided by dynamic delta encoding. Optimized delta encoding approaches that of the dynamic algorithm, but it has issues related to programmability and deployment. It is impractical to individually characterize and program every sensor location and every axis, especially since that process will likely have to be performed for every BSN platform, application, and wearer. Dynamic delta encoding provides better compression while enabling the same algorithm to be pervasively implemented.

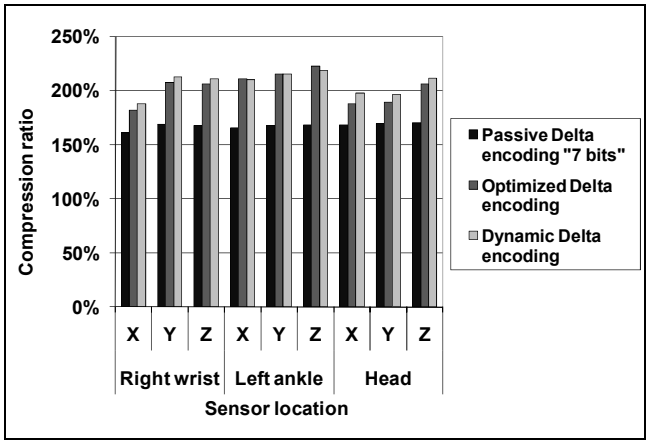

Figure 6. Compression ratios for delta encoding

While neither of these alternative delta encoding techniques benefit from dynamic adaptation to activities, they are both simpler algorithms (no dynamic decision making and no need to keep track of the number of overflows) and require fewer processing cycles and less memory as a result. The average number of processor cycles per sample on the MSP430F1611 to execute dynamic delta encoding was about 95, while the two static techniques (passive and optimized) required only 75.

Figure 7 shows the percent energy savings provided by the three delta encoding techniques over the uncompressed baseline. As specified in Equations 1 and 2, these results take both transmission energy and processor energy into account. The transmission energy per bit and processor energy per active cycle were taken from the datasheets of the Bluetooth module [3] and the MSP430F1611 [11], respectively. It is interesting to note that when processor energy is taken into account, the optimized delta encoding sometimes provides slightly higher energy savings than the dynamic algorithm. However, the programmability and deployment issues remain and prevent optimized delta encoding from being practical for most BSN applications. Dynamic delta encoding provides nearly the same energy savings while being much easier to program and deploy.

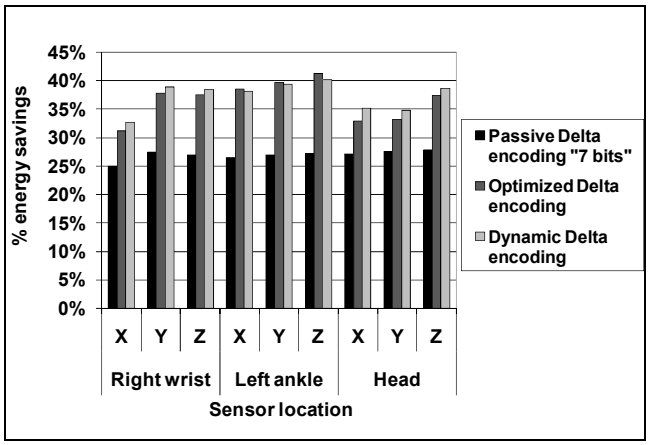

Figure 7. Percent energy savings for delta encoding

7. CONCLUSION

Given that the vast majority of energy consumption in most BSN platforms is due to the wireless transmission of sensed data, pre-transmission data compression is one of the most direct and high-impact ways to increase the energy efficiency of BSNs. However, the use of compression comes with tradeoffs and constraints, especially given the extreme resource limitations on BSN nodes and the many static and dynamic variables associated with various BSN applications, wearers, sensor locations, sensor axes, dynamic activities, etc. First, the compression algorithm must fit within the limited memory of a BSN embedded processor while providing real-time performance. Second, given the difficulty of differentiating important from unimportant data and the criticality of many target BSN applications (e.g. those in the medical domain), lossless compression is desirable and can be made application independent. Finally, in order to provide significant energy savings, the algorithm must maintain a high compression ratio regardless of the static and dynamic variable settings and without significant additional programming and deployment effort. That is, it is highly desirable to program every sensor node the same way without profiling and without consideration of application, sensor location, etc.

This paper evaluated two families of lossless compression techniques – Huffman encoding and delta encoding – within the context of these BSN requirements using a custom accelerometer-based BSN platform within a motion capture application. Both static and adaptive/dynamic variations of these techniques were considered. Results revealed that dynamic delta encoding provided the best combination of energy savings (including both reduced transmission energy and increased processing energy), low memory requirements, high performance across a range of activities and sensor locations/axes, and low programming and deployment effort. The approximately 35% average energy savings provided by dynamic delta encoding can go directly towards extending a BSN platform's battery life and/or reducing the required battery (and total BSN node) size. The results for compression ratios and the energy savings depend on the platform's processor and wireless communication system. However, the trend of these results and the analysis of the other metrics can be generalized to other platforms, applications, and dynamic data.

While dynamic delta encoding showed strong adaptability, future work will evaluate its performance on other BSN platforms and

Digital Object Identifier: 10.4108/ICST.BODYNETS2009.6017
http://dx.doi.org/10.4108/ICST.BODYNETS2009.6017

applications and in combination with other on-node signal processing techniques, such as feature detection and pattern classification algorithms.

8. ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their helpful suggestions and the INERTIA group at UVA for their support and equipment. This work is supported in part by the National Science Foundation under grant Nos. CBET-0756645 and IIP-0646008, the ARCS Foundation, Philips Research North America, and the University of Virginia Institute on Aging.

9. REFERENCES

[1] Arici, T., Gedik, B., Altunbasak, Y. and Liu, L., "PINCO: A Pipelined In-network Compression Scheme for Data Collection in Wireless Sensor Networks," *ICCCN Computer Communications and Network,* pp. 539-544, 2003.

[2] Barr, K.C. and Asanović, K., "Energy-aware Lossless Data Compression," *ACM Transactions on Computer Systems*, vol. 24, no. 3, pp. 250-291, 2006.

[3] Bluetooth RN-41 Class datasheet http://www.rovingnetworks.com/documents/RN-41.pdf

[4] Chang, H.L., Yuan, X. and Wolf, W., "LZW-based Code Compression for VLIW Embedded Systems," *Proceedings of Design, Automation and Test in Europe Conference and Exhibition,* vol. 3, pp. 76-81, 2004.

[5] Chou, J., Petrovic, D. and Ramachandran, K., "A Distributed and Adaptive Signal Processing Approach to Reducing Energy Consumption in Sensor Networks," *INFOCOM Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 2, pp. 1054-1062, 2003.

[6] Ganesan, D., Greenstein, B., Perelyubskiy, D., Estrin, D. and Heidemann, J., "An Evaluation of Multi-resolution Storage for Sensor Networks," *ACM Proceedings of the International Conference on Embedded Networked Sensor Systems,* pp.89-102, 2003.

[7] Kimura, N. and Latifi, S., "A Survey on Data Compression in Wireless Sensor Networks," *International Conference on Information Technology: Coding and Computing,* vol. 2, pp. 8-13, 2005.

[8] Lu, W. and Gough, M., "A Fast-adaptive Huffman Coding Algorithm," *IEEE Transactions on Communications*, vol. 41, no. 4, pp. 535-538, 1993.

[9] Mathur, G., Desnoyers, P., Ganesan, D., and Shenoy, P., "Ultra-low Power Data Storage for Sensor Networks," *ACM International Conference on Information Processing in Sensor Networks,* pp. 374-381, 2006.

[10] Mogul, J.C., Douglis, F., Feldmann, A., and Krishnamurthy, B., "Potential Benefits of Delta Encoding and Data Compression for HTTP," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 4, pp. 181-194, 1997.

[11] MSP430F1611 datasheet http://focus.ti.com/lit/ds/symlink/msp430f1611.pdf

[12] Panda, P.R., Dutt, N.D., and Nicolau, A., "On-chip vs. Off-chip Memory: The Data Partitioning Problem in Embedded Processor-based Systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 5, no. 3, pp. 682-704, 2000.

[13] Petrovic, D., Shah, R., Ramchandran, K., and Rabaey, J., "Data Funneling: Routing with Aggregation and Compression for Wireless Sensor Networks," *IEEE International Workshop on Sensor Network Protocols and Applications,* pp. 156-162, 2003.

[14] Powell Jr., H.C., Hanson, M.A., and Lach, J., "A Wearable Inertial Sensing Technology for Clinical Assessment of Tremor," *IEEE Biomedical Circuits and Systems Conference,* pp. 9-12, 2007.

[15] Puccinelli, D. and Haenggi, M., "Wireless Sensor Networks: Applications and Challenges of Ubiquitous Sensing," *IEEE Circuits and Systems Magazine*, vol. 5, no. 3, pp. 19-31, 2005.

[16] Sadler, C.M. and Martonosi, M., "Data Compression Algorithms for Energy-constrained Devices in Delay Tolerant Networks," *ACM proceedings of the international conference on Embedded Networked Sensor Systems*, pp. 265-278, 2006.

[17] Baek, S. J., Veciana, G. D. and Su, X., "Minimizing Energy Consumption in Large-scale Sensor Networks through Distributed Data Compression and Hierarchical Aggregation," *IEEE Journal on Selected Areas in Communications*, pp. 1130-1140, 2004.

[18] Vitter, J.S., "Design and Analysis of Dynamic Huffman Coding," *Annual Symposium on Foundations of Computer Science*, pp. 293-302, 1985.

[19] Welch, T.A., "A Technique for High-Performance Data Compression," *IEEE Computer*, vol. 17, no. 6, pp. 8-19, 1984.

Digital Object Identifier: 10.4108/ICST.BODYNETS2009.6017
http://dx.doi.org/10.4108/ICST.BODYNETS2009.6017