An on-line piecewise linear approximation technique for wireless sensor networks

Eugen Berlin and Kristof Van Laerhoven
Department of Computer Science
Technische Universität Darmstadt
Emails: {berlin,laerhoven}@ess.tu-darmstadt.de

***Abstract*—Many sensor network applications observe trends over an area by regularly sampling slow-moving values such as humidity or air pressure (for example in habitat monitoring). Another well-published type of application aims at spotting sporadic events, such as sudden rises in temperature or the presence of methane, which are tackled by detection on the individual nodes. This paper focuses on a zone between these two types of applications, where phenomena that cannot be detected on the nodes need to be observed by relatively long sequences of sensor samples. An algorithm that stems from data mining is proposed that abstracts the raw sensor data on the node into smaller packet sizes, thereby minimizing the network traffic and keeping the essence of the information embedded in the data. Experiments show that, at the cost of slightly more processing power on the node, our algorithm performs a shape abstraction of the sensed time series which, depending on the nature of the data, can extensively reduce network traffic and nodes' power consumption.**

***Keywords*-sensor data abstraction, piecewise linear approximation, SWAB, time series shape analysis, wireless sensor networks**

I. Introduction

Through recent advances in computer technology, in hardware as well as in software, wireless sensor networks have shown to be easily scalable and deployable for longer periods of time. Deploying a sensor network has various positive well documented implications, such as minimizing the intrusion and disruption of the environment and its inhabitants that are the focus of scientific interest and research. Wireless sensor networks currently have moved away from being just data samplers; monitoring and processing/analyzing slow-moving environmental values such as humidity, temperature or air pressure on the sensor itself have become common practice.

More recently however, sensor networks are also deployed to monitor or detect critical events, such as human activity [1], volcano activity and eruptions [2] or emergency scenarios [3], [4], that require high-fidelity data analysis in (or close to) real-time. This conflicts with the fact that wireless sensor networks are heavily constrained by their hardware resources. Wirelessly transmitting the raw sensor data to a base station that has the processing capabilities will require high amounts of energy, often resulting in the network nodes to run out of battery power. To prolong the lifetime of the sensor network, wireless communication needs to be reduced. This can be achieved by compressing the sensor data before it is forwarded to the base station for further analysis.

Even when sensors are sampled at relatively high frequencies (e.g., from hundreds of Hertz for inertial sensors up to thousands of Hertz for microphones), data abstraction is still possible. Simple features such as mean, RMS, or signal amplitude can be easily computed, reducing the raw data to a fraction of its original size. Unfortunately, for some applications these features are not descriptive enough.

In this paper, we propose a method that abstracts time series to its basic shape descriptor. Our approach is the reduction of the raw sensor signal by computing a piecewise linear approximation that preserves its shape. Since the computational complexity is higher than computing simple features as mentioned above, our work is aiming at a field where monitored phenomena require more potent features, in particular the shape of the sensed signal. Thus, our work is targeting at applications with the following properties:

- frequency sampling is high in relation to communication bandwidth,
- temporal signal patterns are important to observe and preserve (shape of the signal especially),
- local nodes cannot perform immediate pattern detection and need to forward data to a base station for analysis.

Hereby we use the term "shape of a signal" without a formal definition, relying on how it is used in the Data Mining community (as in "shape matching" [5]).

This paper is structured as follows: In section II we will frame our work amid important related work. Section III is dedicated to our embedded approximation algorithm that is described in detail. Our experimental environment and setup, the chosen parameters and the results will be presented in section IV. Finally, we will discuss our results in section V.

II. Related Work

A good example of the use of features in body sensor networks is the Mercury wearable sensor network platform [6] that is used for on-line analysis of motion data to monitor patients with Parkinson's Disease or epilepsy. The authors of Mercury are aiming at long-term deployment of their wearable sensor network, and thus pay a lot of attention both to battery lifetime and hardware resource constraints. Their approach to handle this high-fidelity data is to compute high-level features (such as mean, RMS, maximum peak-to-peak amplitude, peak velocity, and RMS of the jerk time series) from the raw signal and transmit only the features to the base station, thus

preserving a considerable amount of bandwidth and energy. On the other hand, when the features are indicating a motion that is of particular interest, the raw data that was previously stored on the sensor node needs to be downloaded for more detailed analysis. By preserving the shape of the signal, our method would avoid the storage and transmission of raw data as well, this way reducing battery power consumption. Although the characterization of Parkinson's tremors is likely more efficiently done with these high-level features, epileptic seizures would benefit from shape-based representation.

Early work [7] already identified that it is more beneficial to process/compress data on a sensor node and only wirelessly transmit its compression over the network. Various lossless compression algorithms, such as LZW, bzip2 or GP-zip have been evaluated, whereby the authors not only compared the algorithms' compression performance, but also the power consumption needed to compute the compression. While the paper is focusing on the lossless compression of data, our approach targets at other applications where lossy abstraction of data is allowed and even welcome. Still, the basic idea of reducing the amount of data by using relatively cheap CPU computation instead of power-hungry wireless transmission of uncompressed data holds for both approaches.

Run-length encoding (RLE) [8] is a very common and widely used method to compress data, also in wireless sensor networks. In [9], an adaptation of the common run-length encoding algorithm is presented, the K-RLE algorithm, which in essence is run-length encoding with a threshold $K > 0$. The main advantage of K-RLE is that it not only compresses identical sensor values, but, with a correctly chosen threshold, can be used to filter out noise in the signal. It, obviously, performs very efficiently (in terms of the approximation error) on flat data, or data with flat periods in the signal. On the other hand, if the sensor readings are constantly fluctuating, run-length encoding often results in even more data than the plain raw signal. In this paper, we use both the plain raw data transmission, as well as the K-RLE algorithm, to evaluate against the performance of the algorithm proposed by us.

Another approach to approximate sensor readings has been presented in [10]. Here, a sensor node that is monitoring the development of a physical variable over time can adapt to a time series model among a specified set of candidate models, and then only transmit the model parameters. For example, when monitoring temperature or humidity, it can be described by a linear model with corresponding parameters. Once the currently chosen model and its parameters have been transmitted to the base station, future sensor readings can be predicted, both at the base station, as well as on the sensor node itself. If the sensor readings lie within predefined application-dependent error bounds, wireless communication is not needed. When the new sensor readings exceed the error bounds, the model and its parameters will be updated and transmitted to the sink. This approach is interesting for applications where the sensors can be expected to behave according to a small and fixed set of models, such as temperature or humidity sensors. The drawback is that sensor data with behaviors that are harder to model, that require extremely high number of models or that require constant adjustments of the model either are not feasible or result in a large overhead.

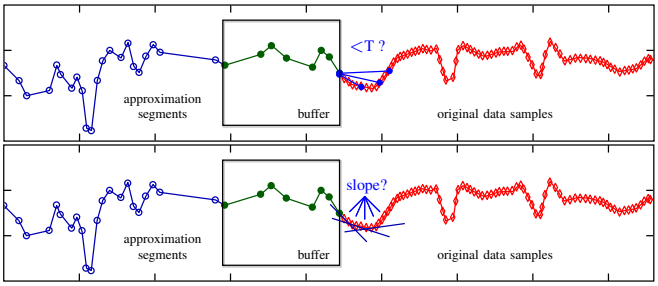

Fig. 1. SWAB approximates a time series to a set of segments, with a sliding buffer in which local bottom-up segmentation is done [13]. Once bottom-up finishes and produces the next segment (here, the leftmost one), the buffer is slid over new data. Our adaptation (below) replaces the original sliding window phase by moving the buffer up to the data point where the slope changes sign.

III. EMSWAB

Key to our approach is the on-line approximation of sampled sensor data into a representation of linear segments that is efficient to manipulate and faster to process than the raw sensor data. The linear segments can be visualized in an identical way to the original data in a time series plot, while the number of data points can be significantly reduced. Our approximation algorithm has its origins in the research field of time series data mining and stems from a modified version of the SWAB algorithm [11], [12] for a wireless sensor node.

A. SWAB

The algorithm proposed in this paper is a modification of SWAB [13] targeting a variety of types of time series data, such as stock exchange data, ECG measurements or powerplant load monitoring. SWAB is a combination of a bottom-up approximation step that is performed on a buffer of raw data points, and the sliding window step where new raw data points are added to the buffer. The authors of the SWAB algorithm mention that optimizations are possible for particular data, for instance, by incrementing the sliding window with multiple samples instead of one (which showed beneficial in case of ECG data). SWAB's standard version moves a sliding window, recalculating an approximation cost and matching it to a threshold for every additional sample of raw data (see top plot in Figure 1).

Our adaptation exploits the property of certain sensors' data, which tend to heavily fluctuate by producing characteristic peaks in the time series, and instead of the sliding window step moves the buffer on to the next data point when the slope's sign changes between positive and negative, or zero (see bottom plot in Figure 1). This means that instead of having to iteratively calculate the approximation cost, one simply has to calculate the slope between adjacent data points x_j and x_{j+1} and stop when the signum function changes value, or $sgn(x_j - x_{j-1}) = sgn(x_{j+1} - x_j)$.

This speeds up the process as it requires a single test per sample ($O(n)$ with n the samples the buffer is shifted over), instead of recalculating costs over the segment ($O(n^2)$ regardless whether sum of squares or the L_∞ norm is used for the cost calculation). Although the bottom-up part of SWAB remains costly, substituting the sliding window technique leads to a significant speedup when the sensors are sampled at a high frequency or if many sub-sequences with a flat signal are present (i.e., when no peaks appear).

Previous performance evaluations of our modified algorithm and the original SWAB have shown that at a marginal increase of the approximation error on a 24-48 hours data set the modified version performs almost twice as fast (~ 1.89). As the original SWAB is known to give a high-quality approximation of the signal and our modified version performs in a similar error range, the remaining question that we will answer in this paper is whether the modified SWAB can be implemented on a sensor network node and at which cost in processing resources.

The next subsection will provide details on the embedded implementation of SWAB's modified version, emSWAB. We will particularly motivate several choices that made the algorithm implementable on a microcontroller-based sensor node.

B. Embedded SWAB (emSWAB)

After implementing the bottom-up approximation as well as the slope sign change part for general purpose personal computers and evaluating their performance on already available off-line data sets, previous experiments indicated that this modification is efficient enough to be implemented on an embedded sensor platform [11], [12]. In this paper we will look at the implementation of this algorithm on wireless sensor nodes in general.

The basic functionality of a sensor unit running emSWAB is as follows: a sensor is being sampled at a specific fixed frequency, producing what we define as raw data values. The new sampled data is forwarded to the emSWAB algorithm that decides whether to store the value to a buffer or, if enough values are available in the buffer already, to run the bottom-up approximation step, thereby producing the next approximating linear segment. This segment is stored to a buffer that eventually will be wirelessly transmitted into the sensor network for further analysis at the base station (see Figure 2). Hereby the segments are represented by data points that consist of an index (Δ_i value in time in amount of samples, to the previous data point) and the corresponding sensor value.

One of the main issues we know in advance we will be facing when porting mSWAB to an embedded system is that a floating point unit is not available in hardware. Thus, following two adoptions were necessary and are important to be noted:

First, the approximation cost function was changed from Euclidean Distance to sum of distances. This way, instead of computing the sum of squared distances and then computing the square root, we only look at absolute distances between the raw data points and their corresponding interpolated value on the segment. This adoption reduces the computational load for

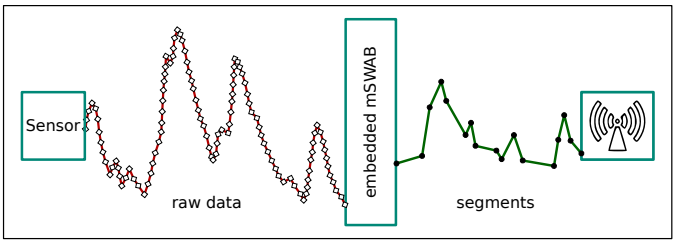

Fig. 2. Basic functionality of emSWAB on a wireless sensor node. The emSWAB takes a buffer of raw data, computes a bottom-up approximation, and produces the next segment as the final result of the approximation step. The buffer is then filled with new raw data. The produced segments are buffered and finally wirelessly transmitted to a base station.

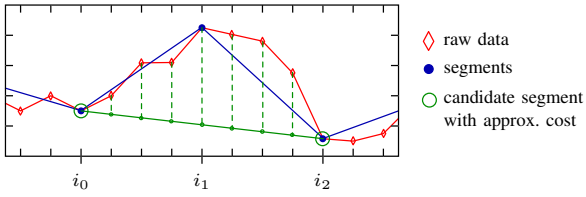

Fig. 3. Example for the computation of the approximation cost (sum of absolute distances) for a candidate segment during the bottom-up approximation step. Since floating point unit is not available on a microcontroller, the computation of the interpolated points on the segment needs additional attention.

the microcontroller, as well as the memory requirements, by avoiding costly implementations of the square and the square-root functions.

Second, when computing the approximation cost between an approximating segment and raw data points, corresponding interpolation points on the segment need to be computed (Figure 3). For example: we want to compute the approximation cost of the candidate segment that will be created when the two blue segments will be merged. Given the two points for the candidate segment $(i_0; v_0)$ and $(i_2; v_2)$, both consisting of an index and the sensor value, we can utilize the linear interpolation formula

$$v_{interp} = v_0 + (i_0 + i_{interp})\frac{\Delta_v}{\Delta_i}$$
$$\text{where } \Delta_v = v_2 - v_0 \ , \ \Delta_i = i_2 - i_0$$

to compute the interpolation values for the indices $i_{interp} \in \{i_0 + 1, i_0 + 2, \ldots, i_1, i_1 + 1, \ldots, i_2 - 1\}$. The approximation cost c is then the sum of absolute distances between the raw data values and the interpolated points:

$$c_{(i_0,i_2)} = \sum_{k=i_0}^{i_2} |v_k - v_{interp,k}| \quad .$$

The lack of the floating point unit on a microcontroller is especially grave for the division step in the formula above that will cause additional error. For example, this error becomes in particular obvious when considering a candidate segment with a positive slope that is still smaller than 1: if $(i_0 + i_{interp}) \cdot \Delta_v < \Delta_i$, the division will result in zero. From this follows that the interpolated point equals the raw data value, and this

results in a distance and thus in the approximation cost of zero.

To preserve as much accuracy as possible, we slightly transform the interpolation formula, forcing the division by Δ_i to be the last step in the computation:

$$v_{interp} = \frac{v_0 \cdot \Delta_i + (i_0 + i_{interp}) \cdot \Delta_v}{\Delta_i} \quad .$$

This transformation has been verified to preserve the accuracy better than the traditional way to compute the interpolation, and therefore was implemented in emSWAB.

Additionally, emSWAB can be further optimized by reducing the amount of repeating computations. For example, when the initial bottom-up approximation costs for a given buffer of raw data have been computed, a copy of those values can be stored and partially reused later. Once the approximation is finished and the next segment is produced, only the costs for those points will be removed that have been merged out by this particular segment. Other raw data points and their corresponding initial approximation costs remain. The approximation costs only need to be computed and stored for new incoming raw data values.

To allow easy portability to different hardware platforms and operating systems, we have chosen to implement our emSWAB algorithm as a library in standard C(95/99). The source code of the library has been made available at the project website[1].

With the emSWAB algorithm now described in detail, in the next section we will present the experiments methodology and evaluate its performance on several data sets of which the data could be captured within a wireless sensor network.

IV. Experiments

In this section we will present the performance of the presented emSWAB algorithm and compare it to two other data dissemination techniques: transmitting the raw data without any compression or abstraction (RAW), and the run-length encoding of the raw data (RLE). The evaluation is done on several representative data sets in wireless sensor network applications.

First, we will present the methodology of the experiments, the data sets on which we evaluate our algorithm, and the experimental setup, and then look at the performance figures under various circumstances.

A. Methodology

The raw sensor data that is used during these experiments was taken from various public data sets. Examples of the data can be seen in Figure 4. Hereby, the data sets ECG 1 and 2, Power and Space are subsequences of the data sets that are freely available at http://www.cs.ucr.edu/~eamonn/discords/. The data sets Sleep and Hapkido are subsequences of data sets that are freely available at http://porcupine2.sourceforge.net/.

The two ECG data sets represent two different anomalies in a normal heart beat, where the first is an oscillation

[1]http://www.ess.tu-darmstadt.de/projects/

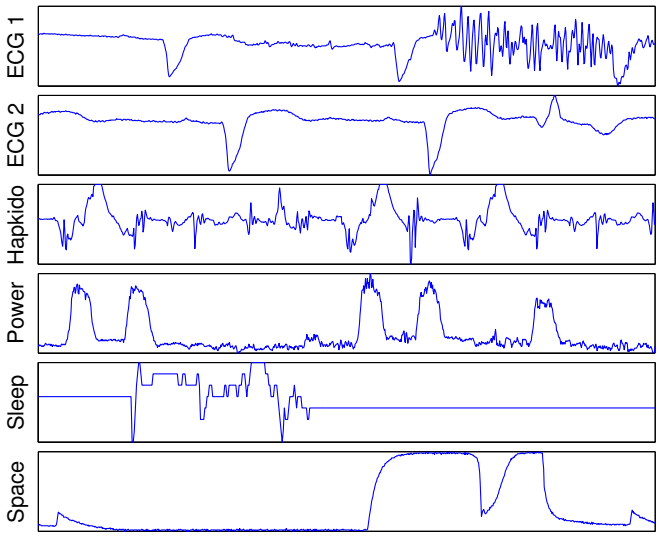

Fig. 4. Different publicly available data sets used in the experiments. The data sets were chosen for their variety of the signal, displaying characteristic patterns. Whenever present, we have chosen the subsequences that contained anomalies, such as high variant section in the latter part of ECG 1 (top plot) or the sudden drop in the "energized" phase of the Space data set.

of high frequency and the second is a small bump. The Power data set represents power requirements/loads that were recorded at a powerplant and spans multiple days. The Space data set shows a normal cycle in a Space Shuttle Marotta Valve time series, with an anomaly during the "energized" phase, where the signal shall stay high. The Hapkido data set shows human activity data, namely acceleration data of a sensor that was attached to the ankle of a person performing Shinson Hapkido training. Finally, the Sleep data set shows accelerometer signals of a person sleeping and changing her sleeping position during the night.

The main aspect about the used data sets are the essentially different types of signal in terms of occurring patterns: on the one side of the spectrum there are long spans with flat signals (as in Sleep or Space) and constantly varying sensor values on the other side (as in Hapkido). Hereby, for our experiments and evaluation, the original sampling frequency is not considered important. More important is the shape of the signal and its preservation during the on-line approximation and abstraction on the sensor node.

We use these data sets to compress the signal utilizing our approach as well as the forwarding of raw data and the run-length encoding technique. With our approach, once a new sensor value is sampled, it is forwarded to the emSWAB algorithm. The resulting approximating segments are buffered first and transmitted via radio to its neighbors or a base station when enough segments have been accumulated. Hereby, the segments are represented by data points consisting of a Δ_i value (number of samples that were abstracted to the previous segment point) and the actual sensor value.

Our evaluation goal is to be able to provide specific performance figures and estimated power consumption for the

algorithm that is easy to reproduce. To achieve this, we have ported the emSWAB algorithm to Contiki [14]. Contiki is a very popular and widely used open-source, multi-platform and multi-tasking operating system for embedded wired and wireless sensor networks. It is written in C and is designed for variety of microcontroller- and microprocessor-based sensor nodes that have limited hardware resources. One of the main aims is a low-power radio communication in wireless sensor node networks. The Contiki OS project has various tools and implemented features, whereby the two most important for our work are the Cooja network simulator [15] and the power-profiling mechanism Energest [16].

Most experiments were conducted using the Cooja cross-level sensor network simulator. As the experiment target system we have chosen the very popular Tmote Sky / TelosB sensor node as the hardware platform. The node is a 8MHz MSP430-based board with 10kB RAM, equipped with a CC2420 IEEE 802.15.4 compliant radio chip, 1 megabyte external flash memory, and optional sensors. The most important power consumption figures, as noted in the Crossbow TelosB data sheet, are:

- MSP430 and circuit power consumption:
  $1.8mA$ in active and $5.1\mu A$ in sleep mode,
- RF transceiver power consumption:
  $23mA$ in active, $21\mu A$ in idle and $1\mu A$ in sleep mode.

Using Contiki's software-based on-line power-profiling mechanism Energest [16], we were able to record how long the sensor node was staying in the low power mode (LPM) or was busy doing some computation (CPU), for example emSWAB computing the approximating segments, or was wirelessly transmitting (TX) data to the base station. Since our experiments focus on the sensor node sending abstracted data only, the time spent for actively listening or receiving data (RX) is negligible. By logging the time spent in each of these four states (modetime) and applying the corresponding power consumptions, current or overall power consumption per state over a period of time can be estimated, by for instance:

$$Power_{CPU}(mW) = \frac{\text{modetime}(ticks) \cdot 1.8(mA) \cdot 3(V)}{\text{frequency}(\frac{ticks}{s}) \cdot \text{runtime}(s)} \quad .$$

Besides being able to use current power consumption to adapt sensor nodes behavior as it is for example done in related work, our primary goal is to evaluate how much power can be preserved by computing a linear approximation of the signal.

B. Some figures on the sensor node images

The footprint of our experimental Tmote Sky / TelosB module image with emSWAB implemented in Contiki is 364.120 bytes, where less than 1% is taken by the emSWAB image. On a different sensor node, featuring a PIC18-based microcontroller from Microchip instead, our emSWAB implementation has a footprint of 2440 bytes.

The buffer size for the bottom-up approximation step was initially set to 20 values, bounding the buffer to minimum 10 and maximum 40 raw data values. When the approximation is computed, a buffer with corresponding indices that are stored as 16 bit unsigned integers is additionally needed. Thus, our implementation results in a buffer of 40 unsigned bytes (sensor values) and a buffer of 40 unsigned 16 bit integers (indices), requiring 120 bytes of memory. To be able to store and work with merging costs for adjacent pairs of segments, an additional array of unsigned 16 bit integers is needed. In order to speed up the costs computation (as described previously), two costs arrays are used, resulting in additional memory requirements of 156 bytes.

The structure implemented as union used to store the segments that are produced during the approximation step is 40 bytes large. Its size can be varied, depending on the size of the wireless communication package. We chose a value of up to 20 data points (19 segments) that are represented by an index and the actual sensor reading.

To avoid losing new raw sensor readings, a raw data buffer of size 120 is used, adding 120 bytes to the overall memory requirements. The buffered sensor readings will be copied into the emSWAB buffer. Once the approximation is computed, the data points that have been merged to one resulting segment will be deleted.

C. Results

After discussing the methodology and presenting some figures on the sensor node images, in this section we want to evaluate the performance of emSWAB for different merging threshold against the raw data dissemination and the run-length encoding techniques.

In Figure 5, we see two plots that visualize the Energest power estimation logs: the upper plot for run-length encoding and the lower plot for emSWAB with a merging threshold of 5 and an initial bottom-up buffer size of 20. Here, the Hapkido data set was used. Note that this is only a fraction of the whole log that was recorded during the simulation, since visualizing the log entirely would make it unreadable. The X-axis represents the sensor reading ticks, whereas the Y-axis shows the number of CPU cycles that were spent in the four different system modes.

Both plots show that most of the time the sensor node stayed in low-power mode. Sampling and storing a new sensor value is almost negligible when abstracting the data with run-length encoding, therefore CPU usage only appears when the data is wirelessly transmitted to the network (MAC overhead). The emSWAB plot, on the other hand, shows that more computation is required. However, the amount of data that needs to be wirelessly transmitted is reduced, resulting in a decreased amount of wireless communications: with the emSWAB approach 4 times, against 7 times with run-length encoding.

Since in the example given above a relatively low merging threshold of 5 has been used for emSWAB, the resulting approximation is close to raw data. Increasing the threshold will result in a more coarse grained approximation. This will lead to more computation on the one hand, thus increasing the CPU load, but will reduce the amount of data to be transmitted even more.

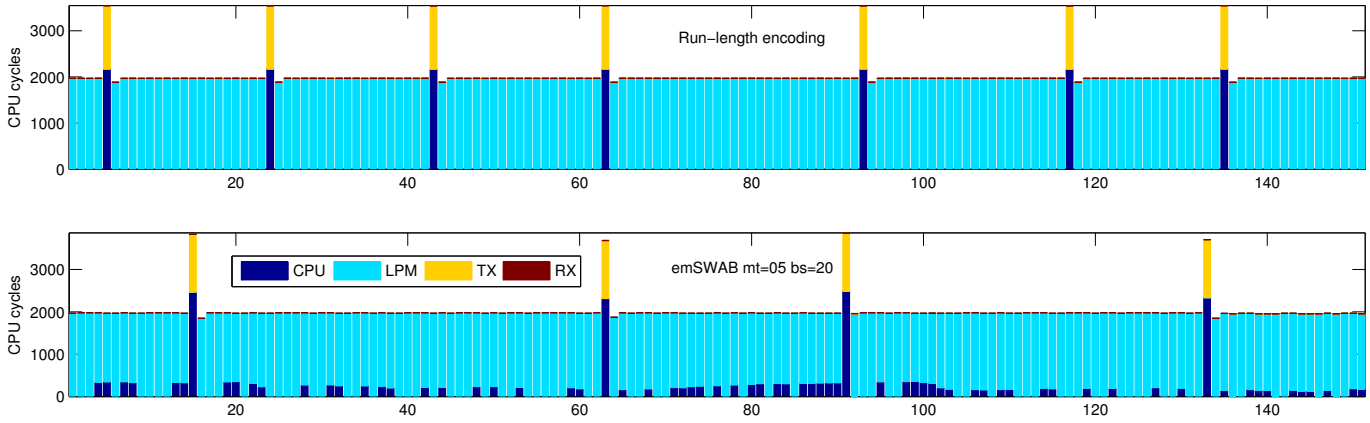

Fig. 5. Power consumption estimation with Energest: comparing the performance of run-length encoding and emSWAB techniques using the Hapkido data set. The plot show how long (in terms of CPU cycles) the system stayed in one of the four modes: CPU - time spent on calculation, LPM - low power mode, TX - wirelessly transmitting the abstraction, RX - listening or receiving (negligible). Run-length encoding performs very similar to raw data dissemination (not shown here). Obviously, the emSWAB algorithm requires more time for computation, but profits through less data that needs to be wirelessly transmitted. Increasing the merging threshold for emSWAB will result in more computation, but due to a more coarse grained approximation in even less data to be transmitted.

In the following, Figures 6 - 11 present emSWAB's performance on the data sets mentioned before (see Figure 4). To be able to compare the performance figures of the different sensor data abstraction techniques, and to decide which one performs better or worse, we need to compute the relative time required for every system mode. To achieve this we sum the time the node has spent in each of the four different system modes, and weigh these by the overall time. Additionally, for every data set emSWAB's merging threshold is varied to show its direct impact on the algorithm's performance and the estimated power consumption. The values used for the merging threshold are $mt = \{5, 10, 15, 20\}$.

Figure 6 shows performance figures for the Hapkido data set. This data set has a highly fluctuating signal that leads to a poor performance of the run-length encoding technique. A fine-grained emSWAB approximation ($mt = 5$) demands more time for its computation, thus resulting in a higher CPU load, but is balanced out by a smaller-sized abstraction. This results in lower package size and thus in less wireless communication, thereby preserving battery power. Increasing the merging threshold and by this forcing a more coarse-grained approximation will reduce the footprint even more, as can be seen in the plot, but result in a higher approximation error that might become critical for preserving the shape.

Figure 7 shows performance figures for the Sleep data set. This data set is different in nature compared to the Hapkido data set, in that it contains long periods with constant sensor values: the signal stays flat, is shortly interrupted by a jump to another level, and then stays flat again. The good performance of the run-length encoding abstraction is therefore not surprising. emSWAB does perform well on this data, too, but at a much higher cost in terms of computation time, preventing the sensor node from entering the desirable low-power mode.

Figures 8 and 9 show performance figures for the two ECG data sets. The main difference in these two data sets

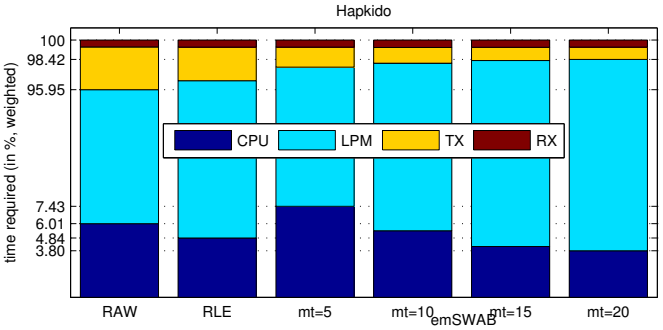

Fig. 6. Energest performance of raw data dissemination, run-length encoding and emSWAB for different merging thresholds using the Hapkido data set. In this data set, the data exhibits continuing varying patterns, which lead to a poor performance of the run-length encoding method. The high computational demand of emSWAB is balanced out by the decreased amount of data to be transmitted.

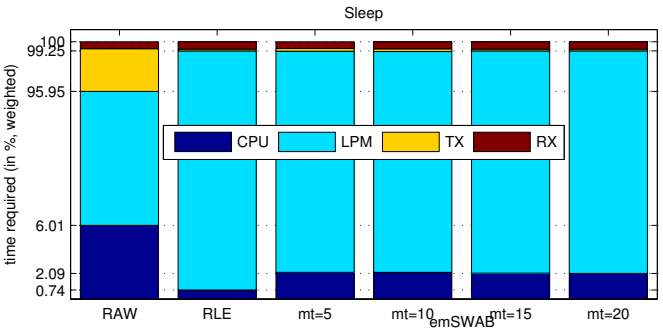

Fig. 7. Energest performance of raw data dissemination, run-length encoding and emSWAB for different merging thresholds using the Sleep data set. For this data, RLE is superior to both raw data dissemination as well as our emSWAB approach. Since sensor values remain the same for long periods of time, transmitting data is minimized when compared to the RAW method. Also, the overhead of emSWAB in processing means that less time is spent in the power-saving mode.

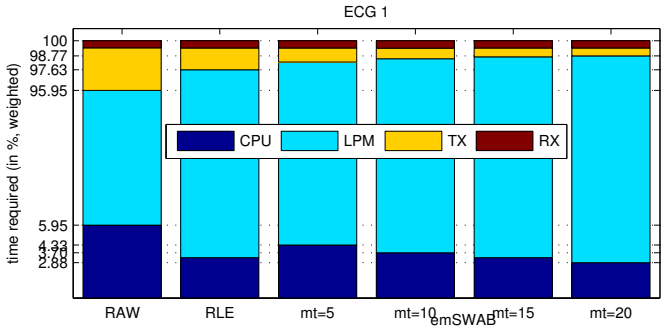

Fig. 8. Energest performance of raw data dissemination, run-length encoding and emSWAB for different merging thresholds using the ECG 1 data set. Since the data set contains flat as well as high variety signal, the overall performance of run-length encoding fits between the Hapkido and Sleep data sets. emSWAB requires additional time for computation, but results again in a lower amount of data to be transmitted, outperforming both RAW and RLE.

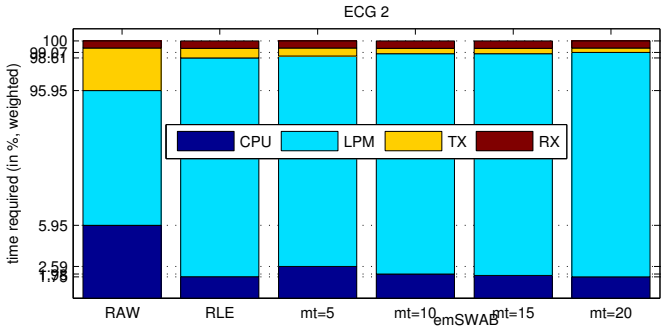

Fig. 9. Energest performance of raw data dissemination, run-length encoding and emSWAB for different merging thresholds using the ECG 2 data set. These results give a very similar picture to the performances that were achieved on the ECG 1 data set. Since the signal in this case lacks the high variance part, the time needed for the emSWAB computations is drastically reduced. This is also the reason for the better performing RLE.

is the type of the anomaly present in the signal. ECG 1 has a part with highly varying signal that prevents run-length encoding to perform as good as it does on ECG 2 that does not contain this kind of anomaly. emSWAB performs well on this combination of flat and high variance signal, outperforming RLE and matching the time needed for computation (and MAC communication overhead) when approximating with a merging threshold of 20. Since the signal in the ECG 2 data set has no high variance part, run-length encoding produces an abstraction of a smaller size. On the other hand, also emSWAB performs better on this data set, again outperforming RLE even for the small merging threshold.

Figure 10 shows performance figures for the Power data set. Due to reoccurring characteristic patterns as well as lots of noise in the signal, run-length encoding performs worse than on clean data as it is the case for Sleep or ECG 2 data sets. In this case emSWAB performs much better, especially with a higher merging threshold. Filtering out the noise and thus considerably reducing the footprint of the approximation, emSWAB even outperforms RLE in terms of cumulative CPU load (approximation computation plus the wireless communication overhead).

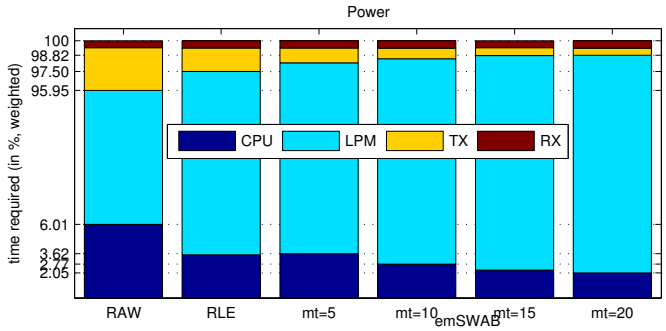

Fig. 10. Energest performance of raw data dissemination, run-length encoding and emSWAB for different merging thresholds using the Power data set. The performance of the run-length encoding technique is significantly worse due to the high level of noise in the signal. In this case emSWAB with a higher merging threshold shows much better performance. This indicates that the noise is filtered out, resulting in a smaller approximation footprint.

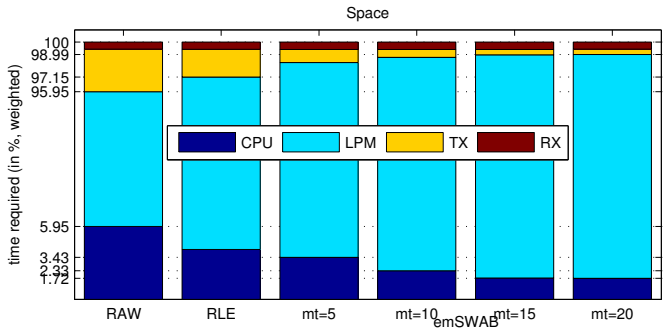

Fig. 11. Energest performance of raw data dissemination, run-length encoding and emSWAB for different merging thresholds using the Space data set. The performance figures on this signal are very similar to those of the Power data set, again due to the signal shape. The data mostly consists of parts where the signal slowly climbs of falls, resulting in a poor abstraction when using the RLE approach. Approximating the signal with linear segments (emSWAB) allows us to preserve the shape of the signal with less data.

Figure 11 shows the performance figures for the Space data set. The signal in this data set contains less noise as in the previous one, but run-length encoding is still not performing optimal. This is due to the underlying shape of the signal that contains long periods of slowly climbing or falling sensor values. emSWAB's linear approximation does preserve this linear shape of the signal, reducing the amount of data to be transmitted, outperforming run-length encoding both in terms of the footprint as well as the time needed for computation and the communication overhead.

In this section we have in detail presented the experiments methodology, some figures on the sensor node images and the Energest performance figures for our emSWAB approximation technique that we have compared to the raw data dissemination as well as the run-length encoding methods. In the next section we want to draw some conclusions from these results and also indicate directions for future work.

V. Conclusions

With the experiments and the results presented in the previous section, we can first conclude that the modification and optimization made SWAB runnable on a microcontroller-based sensor node. Experiments with this implementation of emSWAB, although currently conducted in the Cooja simulator, show good performance on various different data sets.

The comparison of emSWAB's Energest performance figures to the figures of the commonly used run-length encoding technique have shown that emSWAB can provide a good piecewise linear approximation of the signal that preserves its shape and - on most of the data sets used in this study - has a smaller footprint. The abstraction's size is especially crucial to a wireless sensor network as it has a direct impact on the amount of wireless communication in the network and therefore on the battery lifetime.

Additional computation overhead that is needed to produce the emSWAB's approximation is balanced out by the reduced amount of data that needs to be transmitted wirelessly throughout the network. This way, battery power can be preserved much better than with RLE or just transmitting raw data.

Our approach is especially targeting data with patterns essential to the sensor network application. This, of course, does not hold for sensor data that is similar to for instance that in the Sleep data set, as in this case other abstractions such as RLE perform better, both in terms of the abstraction size as well as the CPU load. Future work can consider further optimizations of emSWAB and possible combinations with other abstraction techniques, depending on the signal.

Besides the on-line adaptation of the approximation technique based on the sensor data, current power consumption, power consumption over a time span or the (estimated) remaining battery power can be utilized for adaptive sensor node behavior. For our approach, the adaptive behavior would mean that (based on the power consumption figures) the sensor node will automatically increase or decrease the merging threshold to allow more coarse- or fine-grained approximation of the sensor data. Increasing the merging threshold, thus reducing the number of approximating segments, will result in less data to be wirelessly transmitted and thus preserve battery power. On the other hand, if the approximation needs to be especially fine-grained, and enough battery power is available, the merging threshold could be reduced allowing to capture even fine details in the signal.

Planned experiments are targeting extended sensor network simulations with multiple sensor nodes and corresponding in-network communication as well as real world deployments of wireless sensor nodes with implemented emSWAB in order to confirm the Energest power consumption estimations and to measure the quality of the abstraction.

Acknowledgments

This work was sponsored by the Project "Long-Term Activity Recognition with Wearable Sensors" (LA 2758/1-1) from the German Research Foundation (DFG).

References

[1] M. Marin-Perianu, C. Lombriser, O. Amft, P. Havinga, and G. Tröster, "Distributed activity recognition with fuzzy-enabled wireless sensor networks," in *DCOSS '08: Proceedings of the 4th IEEE international conference on Distributed Computing in Sensor Systems*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 296–313.

[2] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh, "Deploying a wireless sensor network on an active volcano," *Internet Computing, IEEE*, vol. 10, no. 2, pp. 18–25, March-April 2006.

[3] T. Gao, C. Pesto, L. Selavo, Y. Chen, J. G. Ko, J. H. Lim, A. Terzis, A. Watt, J. Jeng, B. rong Chen, K. Lorincz, and M. Welsh, "Wireless medical sensor networks in emergency response: Implementation and pilot results," in *2008 IEEE Conference on Technologies for Homeland Security*, December 2008, pp. 187–192.

[4] T. Gao, T. Massey, L. Selavo, D. Crawford, B. rong Chen, K. Lorincz, V. Shnayder, L. Hauenstein, F. Dabiri, J. Jeng, A. Chanmugam, D. White, M. Sarrafzadeh, and M. Welsh, "The advanced health and disaster aid network: A light-weight wireless medical system for triage," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 1, no. 3, pp. 203–216, September 2007.

[5] E. Keogh, L. Wei, X. Xi, S.-H. Lee, and M. Vlachos, "LB-Keogh supports exact indexing of shapes under rotation invariance with arbitrary representations and distance measures," in *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 2006, pp. 882–893.

[6] K. Lorincz, B. Chen, G. W. Challen, A. R. Chowdhury, S. Patel, P. Bonato, and M. Welsh, "Mercury: a wearable sensor network platform for high-fidelity motion analysis," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*. Berkeley, California, USA: ACM, 2009, pp. 183–196.

[7] K. Barr and K. Asanović, "Energy aware lossless data compression," in *MobiSys '03: Proceedings of the 1st international conference on Mobile systems, applications and services*. New York, NY, USA: ACM, 2003, pp. 231–244.

[8] S. W. Golomb, "Run-length encodings (corresp.)," *IEEE Transactions on Information Theory*, vol. 12, no. 3, pp. 399–401, July 1966.

[9] E. P. Capo-Chichi, H. Guyennet, and J. Friedt, "K-RLE: a new data compression algorithm for wireless sensor network," in *International Conference on Sensor Technologies and Applications*, vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 2009, pp. 502–507.

[10] Y. L. Borgne, S. Santini, and G. Bontempi, "Adaptive model selection for time series prediction in wireless sensor networks," *Signal Processing*, vol. 87, no. 12, pp. 3010–3020, Dec. 2007. [Online]. Available: http://www.sciencedirect.com/science/article/B6V18-4NTJGWG-5/2/33b48c527f2fd2e266a28e1cc63d02d5

[11] K. V. Laerhoven and E. Berlin, "When else did this happen? Efficient subsequence representation and matching for wearable activity data," in *ISWC '09: Proceedings of the 2009 International Symposium on Wearable Computers*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 101–104.

[12] K. V. Laerhoven, E. Berlin, and B. Schiele, "Enabling efficient time series analysis for wearable activity data," in *ICMLA '09: Proceedings of the 2009 International Conference on Machine Learning and Applications*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 392–397.

[13] E. J. Keogh, Chu, Hart, and Pazzani, "An online algorithm for segmenting time series," in *IEEE International Conference on Data Mining*, 2001, pp. 289–296.

[14] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 455–462.

[15] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with cooja," in *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, 2006, pp. 641–648.

[16] A. Dunkels, F. Osterlind, N. Tsiftes, and Z. He, "Software-based on-line energy estimation for sensor nodes," in *EmNets '07: Proceedings of the 4th workshop on Embedded networked sensors*. New York, NY, USA: ACM, 2007, pp. 28–32.