An FPGA Implementation of an Adaptive Data Reduction Technique for Wireless Sensor Networks

Nicholas Paul Borg
Department of Communications and Computer Engineering
University of Malta
Email: nicky.borg@gmail.com

Dr. Carl James Debono
Department of Communications and Computer Engineering
University of Malta
Email: cjdebo@eng.um.edu.mt

Abstract

Wireless sensor networking (WSN) is an emerging technology that has a wide range of potential applications including environment monitoring, surveillance, medical systems, and robotic exploration. These networks consist of large numbers of distributed nodes that organize themselves into a multihop wireless network. Each node is equipped with one or more sensors, embedded processors, and low-power radios, and is normally battery operated. Reporting constant measurement updates incurs high communication costs for each individual node, resulting in a significant communication overhead and energy consumption. A solution to reduce power requirements is to select, among all data produced by the sensor network, a subset of sensor readings that is relayed to the user such that the original observation data can be reconstructed within some user-defined accuracy.

This paper describes the implementation of an adaptive data reduction algorithm for WSN, on a Xilinx Spartan-3E FPGA. A feasibility study is carried out to determine the benefits of this solution.

Index Terms

Wireless Sensor Network (WSN), Power Reduction Technique, Field Programmable Gate Array (FPGA).

1. Introduction

Wireless Sensor Networks (WSNs) consist of a number of spatially distributed, autonomous sensor nodes equipped with sensing, data processing and communications capabilities for monitoring purposes. Since these nodes are battery powered they have a limited supply of energy. However, most WSN applications, for example habitat monitoring and traffic control, require very long periods of operation with minimum or no human intervention. The lifetime of a WSN heavily relies on the existence of power efficient algorithms for the acquisition, aggregation and transmission of sensor readings [1]. Communicating over the radio is the most energy demanding factor. Hence a lot of research is currently underway to develop efficient energy conservation and management techniques.

The main challenges related to WSN implementation are the following [2]:
(1) Energy Conservation – Sensor nodes are battery powered, and therefore contain a limited supply of energy. Moreover the sensor nodes are getting smaller in size, lowering the capacity of the battery even more. Despite this scarcity of energy, the sensor network is expected to operate for a relatively long time. Replacing batteries is most often an impossible task, hence one of the primary design goals is to use this limited amount of energy as efficiently as possible.

(2) Operation in hostile environments and fault tolerance – In extreme conditions sensor nodes must be able to endure harsh environments, thus the protocols for network operation should be resilient to sensor faults which can be considered a relatively likely event.

(3) Hardware limitations – The cost of a sensor network depends on the amount of hardware each sensor node contains. Hence complex hardware should be avoided as this will increase the cost of a sensor network and its up keeping.

The sensor node must be inexpensive such that if it fails, due to insufficient energy or physical damage, it would be more feasible to discard that node rather than replacing its battery pack on site. It must be appreciated that most of the WSN applications require that they work in harsh and hostile environments which are inaccessible to humans.

The aim of a data reduction algorithm is to drastically reduce the amount of reportings done by a sensor node without imposing any hardware constraints which may lead to expensive nodes. Moreover the hardware implementing the algorithm must consume a minimal amount of power. The size of hardware determines the cost of the sensor node and the power consumption. Therefore the algorithm must be small and simple, yet effective and robust.

The data reduction employed was proposed by Santini et al. in [3]. This approach exploits the Least Mean Squares (LMS) adaptive algorithm. This algorithm is used since it provides an important trade-off between complexity and convergence time. It is very simple and it requires a few computations and a small memory footprint. However it provides excellent performance. Moreover, this approach does not require a priori knowledge or statistical modelling of the observed signals. These features make this algorithm versatile and robust to a variety of real-world phenomena. Furthermore, this approach does not require nodes to be assisted by a central entity since no global model parameters need to be defined. Therefore, this scheme can be applied to any network topology (example: star network, clustered network or tree networks) without undergoing any modifications.

This paper first discusses the theory behind the data reduction algorithm. In section 3 a description of the implementation is given while section 4 displays the results obtained. Finally section 5 gives the conclusions made from the results obtained.

2. Theory

2.1. Introduction to Adaptive Filters

Adaptive filters are typically used in applications where the statistical characteristics of the signals to be filtered are either unknown a priori or are slowly time-variant (non-stationary signals) [3]. The generic structure of an adaptive filter is shown in figure 1.

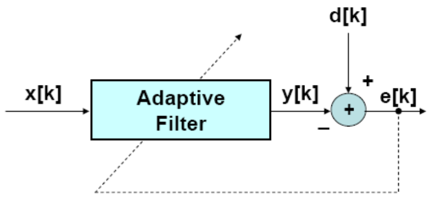

Figure 1. General structure of Adaptive Filter [3].

A linear adaptive filter takes at each step k a sample x[k] of the input sample x and computes the filter output y[k] as:

$$y[k] = \sum_{i=0}^{N-1} w_{i+1}[k] * x[k-i] \qquad \textbf{(1)}$$

The output sample y[k] is a linear combination of the last N samples of the input signal x. Each input signal is weighted by the respective filter coefficient $w_i[k]$. The output signal y[k] is then compared to the reference signal d[k], which is the desired signal the filter tries to adapt to. The error e[k] is the difference between the filter output y[k] and the desired signal d[k].

$$e[k] = d[k] - y[k] \qquad \textbf{(2)}$$

This error is fed into the adaptation algorithm, which accordingly updates the filter weights. The adaptation algorithm modifies the weights at each time step k with the aim to iteratively approach an optimal criterion which is typically the minimization of the Mean-Square-Error (MSE) [3].

2.2. Least Mean Squares Algorithm

Many adaptive algorithms have been developed. The choice of one algorithm over another depends on the trade-off among different factors, including convergence speed, robustness, stability and computational complexity. One of the simplest, yet successful, adaptive algorithm is the Least-Mean-Square algorithm (LMS). The advantage of the LMS is that despite its low computational overhead it provides very good performance in a wide spectrum of applications. The LMS algorithm is defined through three equations that are listed in table 1 [3].

Filter output		$y[k] = \underline{w}^t[k]\underline{x}[k]$
Error signal		$e[k] = d[k] - y[k]$
Weights adaptation		$\underline{w}[k+1] = \underline{w}[k] + \mu\underline{x}[k]e[k]$

Table 1. LMS-Algorithm

where $\underline{w}[k]$ and $\underline{x}[k]$ denote N x 1 column vectors:

$$\underline{w}[k] = [w_1[k], w_2[k], \ldots., w_N[k]]^T \qquad \textbf{(3)}$$

$$\underline{x}[k] = [x[k-1], x[k-2], \ldots., x[k-N]]^T \qquad \textbf{(4)}$$

Parameter μ is the step-size which tunes the convergence speed of the algorithm.

2.3. Prediction Filter

The LMS algorithm can be used to perform prediction when the general filter structure in figure 1 is modified to obtain the predictive structure of figure 2 [3].

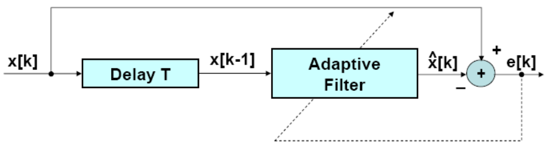

Figure 2. Adaptive filter as a prediction filter [3].

As figure 2 illustrates, the present input value x[k] is delayed by one step and is used as the reference signal d[k]. The filter computes an estimation $\hat{x}[k]$ of the input signal at the step k, as a linear combination of the previous N readings:

$$\hat{x}[k] = \sum_{i=1}^{N} w_i[k] * x[k-i] \qquad \textbf{(5)}$$

The prediction error, e[k], is then computed and fed back to adapt the filter weights. The adaptation process depends on two parameters: the step-size μ and the filter order N. Step-size μ tunes the convergence speed whilst filter order N is a measure of the computational load and memory size of the filter. From equations 3 to 5 it is straightforward to realize that the LMS algorithm requires 2N + 1 multiplications and 2N additions per iteration. Therefore in order to keep the computational load and memory requirements low, the number of weights N must be kept as low as possible.

3. Implementation

3.1 Prediction – Based Monitoring

Figure 3 illustrates two sensor nodes: a data source (A) and a data sink (B). Data source (A) holds a stream of sensor data, {x [k]}, that has to be transmitted to data sink (B). A minimal error budget (accuracy) e_{max} is given and known by both the source and the sink, such that the sink requires to know a value in the range x[k] ± e_{max} rather than the exact value x[k].

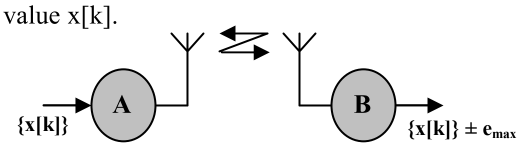

Figure 3. Communication between Source A and Sink B

Instead of transmitting the complete data stream {x[k]} from source to sink, the data reduction algorithm selects a subset of sensor readings that is transmitted to the sink, such that the original observation data, {x[k]}, can be reproduced within the given accuracy.

The data reduction is achieved by introducing identical predictive filters (shown in Figure 2) both in the source and in the sink. The prediction filter produces an estimate of the next sensor reading in the data stream. This estimate depends on the previous sensor readings and on the adaptation weights which the LMS algorithm calculates from the resultant errors. Both the sensor node and the sink apply the same prediction algorithm, hence computing the same prediction of the upcoming reading.

Since the sensor node holds the actual sensor value, it is able to compute the prediction error and compare it with the user-defined error threshold e_{max}. The sensor node only reports the actual value to the sink node when the threshold is exceeded. Otherwise, the sensor node does not transmit its reading. The sink interprets the missing reporting as a confirmation that the predicted sensor value lies within the error budget. Therefore it includes this value in its memory instead of the actual reading. Similarly, the sensor node discards the real measurement and also stores the predicted value. This scheme ensures that at any time instant k, both the sensor node and the sink node share the same knowledge of the observed physical phenomenon.

3.2 Data Reduction using the LMS Algorithm

The LMS algorithm is used for implementing the dual prediction scheme that was described in the previous section. This adaptive algorithm significantly reduces the amount of data that a sensor node is required to report to the sink node, whilst guaranteeing the user-defined error budget e_{max}. The aim of this algorithm is to frequently switch the node's operational mode from its *normal* mode to the *stand-alone* mode. In the latter mode the node does not need to report sensor readings to the sink. In order to be able to run the prediction algorithm, the node needs to go through an *initialisation* phase. These three states of node operation are described below.

(1) Initialization mode: Before performing prediction, both the node and the sink must compute the step-size μ. This parameter is calculated using a set of sensor readings that the sensor node reports to the sink. To ensure convergence, the step-size μ must satisfy the following condition:

$$0 \le \mu \le \frac{1}{E_x} \qquad \textbf{(6)}$$

where E_x is the mean input power computed as:

$$E_x = \frac{1}{M}\sum_{k=1}^{M} |x[k]|^2 \qquad \textbf{(7)}$$

and M is the number of iterations used to train the filter. Since the input mean power E_x is time-varying, an approximation $\bar{E}_x$ can be computed over the first M samples and used to compute the upper bound in inequality (6). It is practical to choose the step-size μ two orders of magnitude smaller than this bound. This guarantees the robustness of the filter. The filter length N is set to very small values that have proven to be efficient for the analyzed data sets. Once the initialization phase is complete, both the node and the sink will start performing prediction on the collected readings and operate in either *normal* or *stand-alone* mode.

(2) Normal mode: Whilst in this mode, both the sensor node and the sink use the last N readings to compute a prediction for the upcoming measurement, and update the set of filter coefficients $\underline{w}$ on the basis of the actual prediction error using equation (8).

$$\underline{w}[k+1] = \underline{w}[k] + \mu\underline{x}[k]e[k] \qquad \textbf{(8)}$$

The initial value of the filter coefficients, $\underline{w}[0]$, is zero. It is imperative that both the prediction filter, located at the sensor node and the prediction filter located at the sink node start with the same initial weights. This guarantees that both compute the same set of filter coefficients and thus, the same predictions, at each time instant k. As long as the prediction error exceeds the user-defined error budget e_{max}, the node keeps working in normal mode, thus collecting and reporting its readings to the sink. Only when the prediction error drops below the threshold e_{max}, does the node switch to *stand-alone* mode. Whilst the sensor node is in *normal* mode, the sink will let the prediction filter run over the received sensor readings, in order to update the filter weights $\underline{w}$ coherently with the node.

(3) Stand-alone mode: Whilst in this state, the node keeps collecting data and computes the prediction at each time step. As long as the prediction error remains below the given threshold e_{max}, the node discards the reading and feeds the filter with the prediction $\hat{x}[k]$ instead of with the real measurement x[k]. This ensures that the state of the filter at the sensor node side remains consistent with the state of the filter at the sink side. During this mode the filter weights are not updated, thus saving half of the computational overhead. Once the sensor node observes that the prediction error exceeds the threshold e_{max}, it will report the reading x[k] to the sink node and switch back to *normal* mode. While the sensor node operates in *stand-alone* mode, the sink, receiving no readings from the node, assumes that the predicted readings lie within x[k] ± e_{max} and keeps running the prediction filter on these values.

3.3. Implementation of the LMS Algorithm on FPGA

The algorithm computes the prediction of the upcoming measurement. The prediction error is then computed and compared with the error budget. As long as the prediction error remains below the given threshold e_{max}, the node discards the real sensor reading x[k] and feeds the filter with the prediction . The filter coefficients (weights) are only updated if the prediction error exceeds e_{max}. The filter order (N) was set to 4, hence the computational cost of the LMS algorithm is 17 when the node operates in normal mode and 8 in stand-alone mode. Moreover this algorithm requires nine memory registers: four 14-bit registers to store the filter coefficients, four 16-bit registers to store the last four sensor readings and one 16-bit register to store the prediction error. The algorithm uses a 22x22 bit multiplier, a 16-bit adder and a 17-bit subtractor. These hardware blocks share the same data bus. The data is routed to the intended hardware by means of a multiplexer. Both the adder and the subtractor were implemented using Carry Look-ahead addition techniques. The multiplier was implemented using the embedded hardware multipliers. The FPGA that was used contains four 18-bit multipliers [4]. All the available multipliers were used to implement the 22x22 bit multiplier. All the computations were carried out using two's complement, fixed point arithmetic. The logic circuits of fixed point hardware are less complicated than those of floating point hardware. This means that fixed point hardware is smaller in size and consumes less power. Moreover fixed point calculations require less memory and less processor time to perform. In order to avoid overflow the fixed-point values were scaled after each computation. The step-size μ was set to a constant of 2^{-13} (1.2207 x 10-4). This fraction can be represented in binary using at least fourteen bits. The filter coefficients (weights) are represented in binary as 14-bit fractions, thus avoiding quantization effects. However the sensor reading contains an integer part which is eight bits. The LMS algorithm requires that sensor measurements are multiplied to weights and/or the constant μ. Since fixed point is used then twenty-two bits (eight bits for integer and fourteen bits for fraction) are required to perform the computations.

4. Testing and Results

The LMS-based data reduction strategy was tested on a set of real world data which is publicly available at [5]. Once every 31 seconds, humidity, temperature, light and voltage values were collected from 54 Mica2Dot sensor nodes [6] that were deployed in the Intel Berkley Research Lab [7] between February 28th and April 5th, 2004. The temperature measurements of four sensor nodes, namely nodes 1, 11, 13, and 49 were used. Six thousand temperature sensor readings for each sensor node, reported between March 6th and March 9th, were applied to the data reduction algorithm.

4.1. Determining Parameters for Data Reduction Algorithm

By illustrating simulation examples, this section describes how the parameters namely the convergence speed μ and filter order N were chosen.

4.1.1. Energy-Accuracy Trade-off

Figure 4 illustrates the percentage of sensor readings that node 49 would need to report as the error budget e_{max} increases. The parameters for filter order N and step-size μ were set to N = 4 and $\mu = 3.0518 \times 10^{-5}$ respectively.

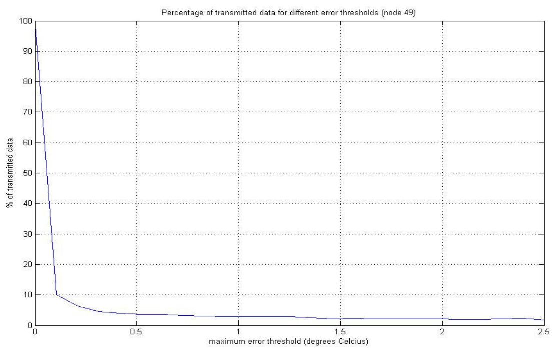

Figure 4. Percentage of data transmitted against error budget e_{max} for node 49.

It is evident that when the error budget is large then less temperature readings are required to be reported to the base station. This implies that more energy is saved, hence increasing the node's lifetime. However, as shown in figure 5, energy is saved at the cost of lessening the degree of accuracy by which the base station can predict the node's temperature measurement.

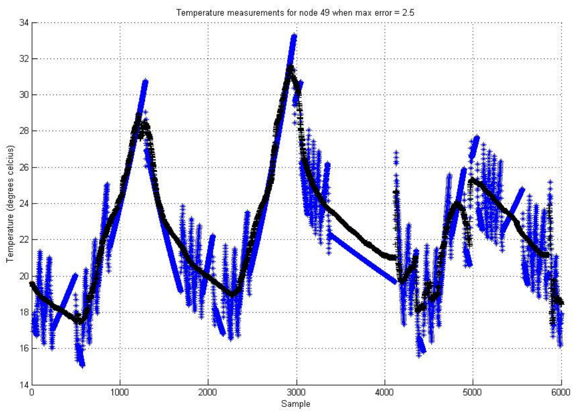

Figure 5. Predicted sensor readings when e_{max} is set to 2.5°C. The blue curve represents the prediction whilst the black curve represents the real measurement.

4.1.2. Filter Order N

No significant changes in the performances were observed when varying the number of filter weights from N = 4 to N = 10. Since the number of operations to be performed at each time step grows proportionately with N (computational cost per iteration of the LMS algorithm is 4N+1 when the node operates in *normal* mode and 2N in *stand-alone* mode), this value should be kept as small as possible. Moreover the size of the memory foot print is strongly related to the filter order. A filter order (N) of four was chosen for this project. Simulations have shown that the performance of the data reduction algorithm deteriorates when the filter order N increases.

4.1.3. Step-size μ

The step-size μ is a critical parameter since it tunes the convergence speed of the algorithm. Choosing a small μ implies fast convergence at the cost of transmitting more sensor readings. A larger μ reduces the convergence speed thus it takes longer for the predicted value to track the real measurement. However a larger μ reduces the percentage of reported sensor readings. If μ is too large the data reduction algorithm will become unstable. Figures 6 and 7 show how the step-size μ affects the performance of the prediction algorithm.

In terms of hardware design this parameter determines the size of the registers inside the

FPGA. Fixed-point arithmetic is used to carry out the calculations. Hence this parameter creates a compromise between accuracy and area. Choosing a small μ will considerably increase the size of the hardware since it takes more bits to represent a fraction.

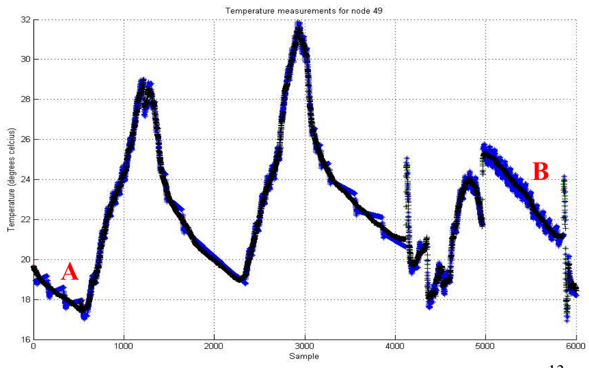

Figure 6. Predicted sensor readings when μ is 2^{-13} (1.2207 x 10^{-4}). Filter order N is 4. The percentage of transmitted readings is 3.633%.

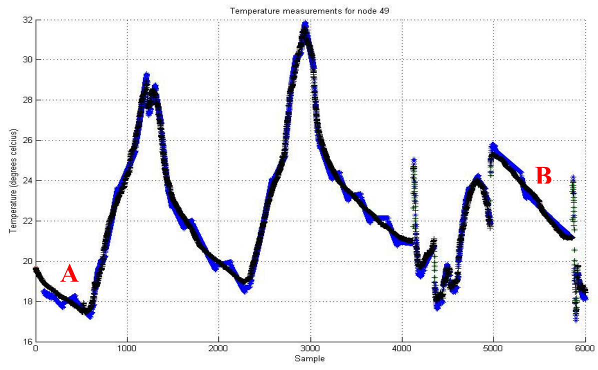

Figure 7. Predicted sensor readings when μ is smaller (3.0518 x 10^{-5}). The percentage of transmitted readings is 4.38%. Note that convergence at A and B is faster when μ is smaller.

In terms of IC manufacturing this signifies a rise in cost. Moreover a larger hardware increases the power consumption thus limiting the lifetime of the device. The data reduction algorithm was simulated with different step-sizes. Table 2 shows the FPGA device utilization for two different step-sizes. The best compromise was obtained when the step-size was set to 2^{-13} (1.2207 x 10^{-4}). This entails that at least 14 bits are required to represent binary fractions. Therefore the total register size amounts to 22 bits (8 bits to hold the integer part and 14 bits to hold the fraction part).

μ	**Hardware muliplier**	**Slices**	**Slice Flip Flops**	**4 i/p LUTs**
2^{-13}	22 bit	239	288	433
2^{-15}	24 bit	250	304	452

Table 2. FPGA device utilization for different μ.

4.2. Results

When applied to a temperature monitoring system, the data reduction algorithm achieved a reduction in transmissions of more than 95% whilst ensuring an accuracy of 0.5 degree Celcius.

4.3. Feasibility Study

In order to physically test the performance of this technique, the algorithm was integrated into a sensor node working on a TDMA protocol. The sensor node was connected to a RF transceiver ER400TRS-02 by LPRS [8]. Two cases were considered, namely:

(1) When the sensor node's transceiver is always switched on and the node transmits all temperature measurements. The expected lifetime of a sensor node running on 2000mA-hr battery-pack was calculated to be 95.2 hours (4 days).

(2) When the data reduction algorithm is integrated into the sensor node. In this case the sensor node is expected to operate for more than 25 months.

5. Conclusion

This work has delved into an adaptive approach that sought to reduce significantly the amount of data that needs to be reported, whilst ensuring that the original observation data is reconstructed within a pre-specified accuracy. The results have shown that the method proposed manages to increase the network life time by 18,962.5 % when compared to an always on solution. This scheme necessitated an increase in size of the IC to contain the necessary hardware blocks that carry out the computations, hence increasing the cost of producing these smart sensor nodes. However, it is envisaged that in the longer term, the benefits will overweigh the initial costs, in terms of a longer life expectancy of the sensor nodes.

6. References

[1] Demetrios Zeinalipour-Yazti, Panayiotis Andreou, Panos K. Chrysanthis, George Samaras, and Andreas Pitsillides: *The MicroPulse Framework for Adaptive Waking Windows in Sensor Networks*, International Workshop on Data Intensive Sensor Networks 2007 (DISN'07), Mannheim, Germany, May 2007.(In conjunction with MDM'07).

[2] P. Santi "Topology Control in Wireless Ad Hoc and Sensor Networks", John Wiley & Sons, Ltd, 2005.

[3] Silvia Santini, Kay Römer: "An Adaptive Strategy for Quality-Based Data Reduction in Wireless Sensor Networks." In Proceedings of the 3rd International Conference on Networked Sensing Systems (INSS 2006). TRF, pp. 29-36, Chicago, IL, USA, June 2006.

[4] Xilinx Spartan-3E FPGA Family, March 2005.

[5] Intel Lab Data. [Online] Available: http://db.lcs.mit.edu/labdata/labdata.html

[6] Crossbow. [Online] Available: http://www.xbow.com/

[7] Intel Berkeley Research Lab. [Online]

Available:http://www.intel-research.net/berkeley/index.asp

[8] Low Power Radio Solutions. [Online] Available: http://www.lprs.co.uk/