AN ENERGY BENCHMARK FOR SOFTWARE UPDATES ON WIRELESS SENSOR NODES

S. Brown*, C.J. Sreenan

*Dept. of Computer Science/Callan Institute, NUI Maynooth, Ireland
email: stephen.brown@nuim.ie

Keywords: WSN, Update, Modelling, Energy, Benchmark.

Abstract

Energy consumption is arguably the key factor in the design and operation of Wireless Sensor Networks (WSNs). This holds both for normal operation and maintenance operations – such as software updates. Whereas software updates will probably be infrequent, they must still not consume a significant fraction of a WSN's energy reserve; also, the required consumption must be known before triggering an update, in order to ensure that it can complete. Software updates are an expensive operation: there can be a significant volume of data, guaranteed delivery is required, and normal data fusion algorithms cannot be used to reduce the communication load. In this paper we present a node-level energy consumption model for the evaluating the efficiency of software updates on wireless sensor nodes. This model is then used to derive a novel minimum energy benchmark equation. This paper also presents some new power measurements, and uses these, along with published data, to interpret the benchmark in quantitative terms for some specific hardware platforms. This benchmark provides a standardized and quantitative figure to use in comparing software update algorithms. The methodology is also applicable to establishing energy benchmarks for other tasks in the WSN domain.

1 Introduction

Energy consumption is arguably the key factor in comparing different algorithms for use in Wireless Sensor Networks (WSNs) [17,18]. Ultimately, energy is consumed by the node hardware – and this sets a lower limit on the energy consumption of any algorithm. Design of progressively more efficient hardware has been a characteristic feature of research into WSNs [8]. But the energy consumed also depends on the system software that runs on a node: the operating system, device drivers, and networking stack. For example, disabling the receiver [20] during overheard/unwanted radio messages, or using different modes of operation [22] allows the hardware to be used more efficiently.

At a higher level than the individual nodes, is the entire network, and the distributed algorithms that run on this. Again, the energy efficiency of the network as a whole depends on how effectively these distributed algorithms use the underlying node hardware and software.

In this work we examine the energy consumed by the various components of a software update operation and build a general model to describe the total energy used. This model is then used to derive a minimum energy equation which can be used as an energy benchmark in evaluating and comparing software update algorithms. Sensor node energy measurements are then presented, and used in the calculation of the minimum benchmark for two similar sensor nodes.

In Section 2 we describe some representative work in WSN software updating, and discuss their energy reduction mechanisms. In Section 3 we show the derivation of our model. In Section 4 we show how this model is used to derive a minimal energy benchmark. Section 5 shows how measurements were made to derive the figures required in Section 6 to calculate actual benchmark figures.

2 Background and related work

Software updating has always been identified as an important topic for wireless sensor network research [15]. As with all WSN mechanisms, reducing and optimizing energy use is a critical concern. Identified requirements include low overheads, and resource awareness – especially to minimize flash rewriting [10]; minimizing the impact on sensor network lifetime, and limiting the use of memory resources [21]; minimizing processing, limiting communication to save energy and only interrupting the application for a short period while updating the code [3,16]; operating within the hardware constraints of different platforms [12].

Power measurements for WSNs have been used to extend the capabilities of simulators (for example [19]), and work on energy evaluation[4] uses a mixture of theoretical, simulation, and real-world results to evaluate energy efficiency. But these results do not provide an absolute benchmark against which to evaluate protocols. Many of the protocols and algorithms developed for WSNs include specific provision for optimizing energy performance, and we consider the energy related aspects of a few representative examples here. A fuller treatment of software updating in wireless sensor networks is available in [1].

Deluge [6] is a data dissemination protocol and algorithm for propagating large amounts of data throughout a WSN using incremental upgrades for enhanced performance. It is particularly aimed at disseminating software image updates, identified by incremental version numbers. The same image is disseminated to all nodes in the network. The program image is split into fixed size pages, and each page is split into fixed size packets to suit the packet size of the TinyOS[7] network stack. A bit vector of pages received can also fit in a single packet. Nodes broadcast advertisements containing a version number and a bit vector for any new pages received, using a variable period based on updating activity. To upgrade part of its image to match a newer version, a node listens to further advertisements for a time, and then requests the page number/packets required from a selected neighbour. A sender collects several requests before selecting a page and broadcasting the requested packets. When a node receives the last packet required to complete a page, it broadcasts an advertisement before requesting further pages - this enhances pipelining of the update within the network. State data takes a fixed amount of space, independent of the number of neighbours. There are no ACKs or NACKs - requesters either request new pages, or missing packets from a previous page. Radio network contention is reduced through heuristics used to select more remote senders. Rateless Deluge and ACKless Delugs [5] improve on this work by using rateless codes to reduce the need for rebroadcasts, and FEC to reduce the number of control packets.

Imapla[11] is the event-based middleware layer of the ZebraNet wireless sensor network. It is designed to allow applications to be updated and adapted dynamically by dispatching events through a application adapters. ZebraNet nodes are expected to be inaccessible, and deployed in large numbers, so ZebraNet supports high node mobility, constrained network bandwidth, and a wide range of updates (from bug fixes, through updates, to adding and deleting entire applications). Applications consist of multiple, shareable modules, organized in 2KB blocks. The Application Updater allows applications to continue running during updates, and can process multiple contemporaneous updates; version numbering is used to ensure compatibility of updates with existing modules. It also handles incomplete updates, and provides a set of simple sanity checks before linking in a new module. Software updates are performed in a three-step process: firstly the nodes exchange an index of modules, then they make unicast requests for updated modules, and finally they respond to requests from other nodes. An exponentially increasing backoff timer reduces management traffic, but can delay updates when separated groups of nodes reconnect. When software reception is complete, then after performing simple sanity checks, the old version application is terminated, the modules in the new version are linked in, and the new application is initialised prior to use.

MNP [23] is targeted at nodes running TinyOS and the XNP boot loader [9]. The protocol operates in four phases. During Advertisement/Request sources advertise the new version of the code, and interested nodes make requests. Sources listen overhead all other advertisements and requests - a suppression scheme to avoid network overload. During Forward/Download a source broadcasts a StartDownload message to prepare the receivers, and then sends the program code a packet at a time - there is no ack. During Query/Update the source broadcasts a Query to all its receivers, which respond by unicast asking for the missing packets. Receivers, having received the full image, now become source nodes and start advertising. During Reboot, entered when a source receives no requests in response to an advertisement, the new program image is transferred to program memory, and the node reboots with the new code. Download requests are sent to all sources to reduce the hidden terminal effect, and select only one active sender in a neighborhood. Flow control is rate based, determined by the EEPROM write speed.

The selected algorithms described above use various techniques to reduce energy use, but there is no standardized method to compare them from an energy viewpoint. In the next sections we present a new model and benchmark approach that does provide a standardized baseline for these energy comparisons.

3 The model

WSN software upgrades require energy for:

1. receiving the upgrade (including retransmissions),
2. transmitting necessary packets to initiate the upgrade and cause retransmissions,
3. processing the upgrade (e.g. security, decompression, dynamic linking), and
4. writing the upgrade to permanent program memory.

The energy used during a software upgrade is the sum of these four factors:

$$E_{total} = E_{receive} + E_{transmit} + E_{process} + E_{store} \quad (1)$$

Note that, in this work, we are *not* considering the cost of distributing the upgrade throughout the network (this is addressed in the section on future work). We also are not considering the cost of writing the upgrade to *temporary* storage: we argue that it is not in general necessary to do this, though it may be on some specific hardware platforms, or used with particular algorithms. Finally we are not considering the energy required to restart or reboot the software: in general this is insignificant, but further work is needed to investigate this.

In the rest of the paper we are using the term *energy* in the general sense. All equations are based on *electrical charge* (measured in micro-Amp-seconds) to represent the energy used. To compare different algorithms on the same platform, this provides an accurate relative measure. To calculate the electrical energy (in Joules) requires a voltage factor.

The formulae for modeling the energy consumed by a software upgrade operation are shown in Equations (1) through (6) – note: all charge is expressed in micro-Amp-seconds. See Table 1 for a description of the equation parameters.

Name	**Description**	**Units**
$E_{receive}$	Energy consumed for the wireless reception of the upgrade	J
$E_{process}$	Energy consumed for the processing of a software upgrade	J
E_{store}	Energy consumed for the storage of a software upgrade into permanent program memory	J
$E_{restart}$	Energy consumed for the restarting of the software following an upgrade (e.g. a reboot)	J
S	Size of the upgrade (in Bytes)	Bytes
N	Number of processing operations required for a software upgrade	Instructions
f_i	Fixed overhead for processing operation *i*	Instructions
v_i	Variable overhead for processing operation *i*	Instructions /Byte
E_{inst}	Energy consumed by executing a single instruction (a single, average energy figure is used as a simplification here)	uAS (micro-Amp-Seconds)
k	Wireless packet payload	Bits
H	Wireless packet overhead (preamble, header, checksum, etc.)	Bits
P	The average bit-error-rate	Errors/Bit

Table 1: Energy Equation Parameters

The receive energy $E_{receive}$ is the product of the energy used per bit (E_{bit}), the bits per packet ($k+h$), and the number of packets received $\frac{(8*S)}{k}$ including a factor to account for retransmissions: $\frac{1}{(1-\rho)^{(k+h)}}$ [23].

$$E_{receive} = E_{bit} * (k+h) * \frac{(8*S)}{k} * \frac{1}{(1-\rho)^{(k+h)}} \quad (2)$$

The transmit energy $E_{transmit}$ is the product of the energy used per bit (E_{bit}), the bits per packet transmitted (S_{tx}), and the number of packets transmitted (N_{tx}).

$$E_{transmit} = E_{txbit} * S_{tx} * N_{tx} \quad (3)$$

The processing energy $E_{process}$ is the sum of the energy of each processing operation; for each operation, the energy is the product of the average per-instruction cost (E_{inst}) and the sum of the fixed cost (f_i) and the variable per-byte cost ($S*V_i$).

$$E_{process} = \sum_{i=1}^{N} \left(f_i + (S * v_i)\right) * E_{inst} \quad (4)$$

The storage energy E_{store} is the product of the number of bytes (S) and the per-byte permanent storage cost (E_{perm}).

$$E_{store} = S * E_{perm} \quad (5)$$

The following characteristic parameters must be measured for a specific hardware platform: $N, f_i, v_i, E_{inst}, E_{perm}$.

Based on this energy equations (1)-(5), we can now derive an expression for the minimum energy required on a node to perform a software upgrade. Note that the form of the equation presented in this paper relates to energy required locally on the node to receive and process the upgrade, and not to the energy required network-wide to distribute the upgrade throughout the network.

4 The minimum energy benchmark

The minimum energy benchmark for a particular platform (reflecting the minimum energy required) is derived by calculating the minimum value for the energy consumption each of the energy consumption Equations (1)-(5), as shown in Equation (6).

$$E_{benchmark} = E_{receive}^{\min} + E_{transmit}^{\min} + E_{process}^{\min} + E_{store}^{\min} \quad (6)$$

This minimum energy level may not be achievable by any particular algorithm, but it sets a quantified minimum baseline below which no algorithm can go: thus it acts as an objective and standardized benchmark both for evaluating a particular software update mechanism for the potential to reduce energy consumption, and for comparing software update mechanisms for to determine their energy efficiency on an absolute scale.

The treatment of retransmissions is a good example of the philosophy behind the minimum energy benchmark approach: for reliable data reception, any missing packets must be retransmitted – but there is no particular algorithm that optimally minimizes the energy for control traffic (e.g. ACKs, NACKs, or other feedback mechanisms), as there is always a time/energy tradeoff to be made. Therefore, the minimum (and unachievable) power benchmark energy figure for the retransmission control traffic is zero – the energy figures for the retransmitted data are of course included.

The (locally) optimum values for the parameters *k* and *h* are determined by the hardware platform (wireless hardware parameters) and the bit-error-rate ρ. The optimal value for *k* in the presence of bit errors, k_{opt}, can be calculated as shown for example in [2], but in this paper we use typical packet sizes.

In the rest of this paper we will use a simplified version of the benchmark – see Equation (7) – assuming no significant processing. The same way as retransmissions overheads are handled by using an ideal zero cost, we also assign an ideal zero cost to the base processing for a software update (i.e. iniating the data transfer from RAM to Flash). Energy costs associated with compression and/or dynamic linking need to be considered separately; the following assumes a *simple* upgrade.

$$E_{benchmark} = E_{receive}^{\min} + E_{transmit}^{\min} + E_{store}^{\min} \qquad (7)$$

This represents the simplest case where there is no processing required for a software update: e.g. where $E_{process}^{\min} = 0$.

For this benchmark, the minimum number of feedback packets required (either to request the upgrade, or to request retransmissions) is taken to be 1 packet.

4 Energy measurement results

To demonstrate the use of the minimum energy benchmark we have picked two wireless sensor nodes based on the same processor (ATMEL ATMega128L) and used energy measurement data to determine value for the required parameters in the energy equation.

Energy measurements for the MICA2 node [2] have been presented in a number of publications, we use the figures from [19] to estimate the values in Table 2. Note that the energy parameters include the CPU executing during receive and transmit operations. The Flash access energy is estimated from the DSystem25 results (see Table 4) and the MICA2 figures.

Parameter	**Value [mA]**
E_{perm}	0.581 uAs/byte
E_{rxbit}	0.539 uAs/bit
E_{txbit}	0.420 uAs/bit

Table 2: MICA2 energy parameters

Current measurements were made on the DSystem25 Module [14], and from these the energy consumption parameters for this platform were calculated. This node contains the following components that use energy during operation:

- Atmel ATMega128L processor;
- a Nordic nRF2401 radio transceiver;
- an LP2966 3.3V regulator;
- a 4 Mhz external oscillator (part IQX0-71).

The measurements were made using a specially developed program running directly on the hardware, using the AVR C library, but no operating system. This allows the hardware-specific energy costs to be measured without the effects of any operating system algorithms. Radio transmission was using 32-byte packets in *Shockburst* mode at 250Kbps.

The current consumption of the module was measured by placing a 1-ohm resistor (±5% of its nominal value) in series with the power supply for the module: a power-measurement methodology for wireless sensor nodes described in other research publications [10]. The power measurements were performed using a Thandor TS1541S power supply providing 6.00V (±0.005V). Some sample results were compared against two new 3v batteries (CR2430) to ensure that the power supply was not introducing anomalies. No significant differences were seen, and so all measurement presented in this paper were made using the lab power supply to guard against voltage drops as the batteries are drained during multiple measurements.

The measurements were made at a sample rate of 1 Mega-samples/second with 16-bit precision using an PCI-6251 NI-DAQ card from National Instruments, with the input voltage configured in the range of +100mV to -100mV. A higher frequency oscilloscope (20MHz) was used to ensure that the 1 MHz sampling rate was sufficient.

The raw measurements show current oscillations (see Fig. 2 for an example). A smoothing capacitor was not used in order to avoid any distortion to sharp transitions in the graphs. Unsmoothed graphs are presented here: the average current consumption and time for each operation was calculated in order to derive the figures shown in Table 3.

A dedicated 'power-signature' program was used that performed the various operations to allow the current consumption (and thus the energy parameters) of each operation to be determined. A companion node was used to provide suitable wireless traffic for reception.

Screenshots of the current measurements are provided in Figs. 1-8 to give the reader an overview of the large data-sets:

- Flash memory erase & write (Fig. 1) – this shows the operation for 1 Flash memory page, 256 bytes
- EEPROM read & write (Fig. 2) – two erase & write operations: of 1 byte followed by 2 bytes
- Wireless transmission (Figs. 3-6) – these show transmission at -20, -10, -5, and 0 dBm. The data transfer, and transmission phases use 28-byte payloads (for a total of 33 bytes on the air including preamble)
- Wireless reception (Fig. 7) - with the receiver enabled, and a single packet received
- CPU executing (Fig. 8) – all peripherals, including the wireless transceiver, are disabled or configured in the lowest available power mode
- CPU sleep mode with all peripherals disabled (Fig. 8)

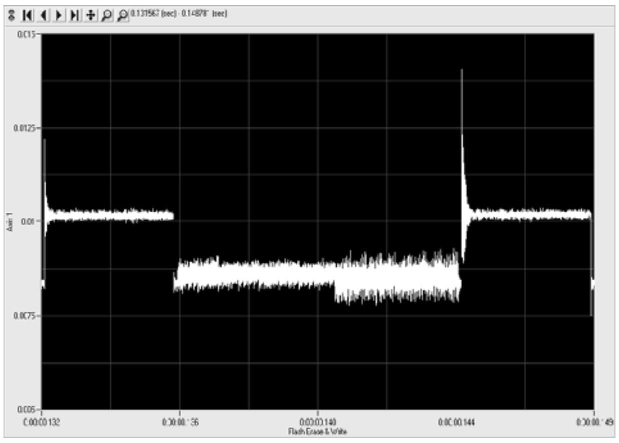
Figure 1: Flash Memory Access current vs Time
Vertical scale: 5mA – 15mA
Horizontal scale: 132mS – 149mS

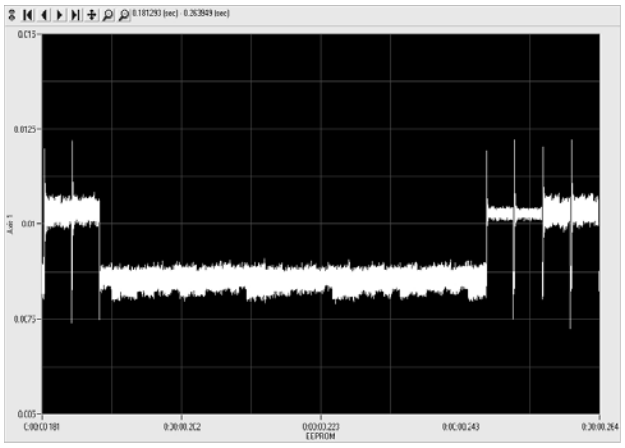
Figure 2: EEPROM Access current vs Time
Vertical scale: 5mA – 15mA
Horizontal scale: 181mS – 264mS

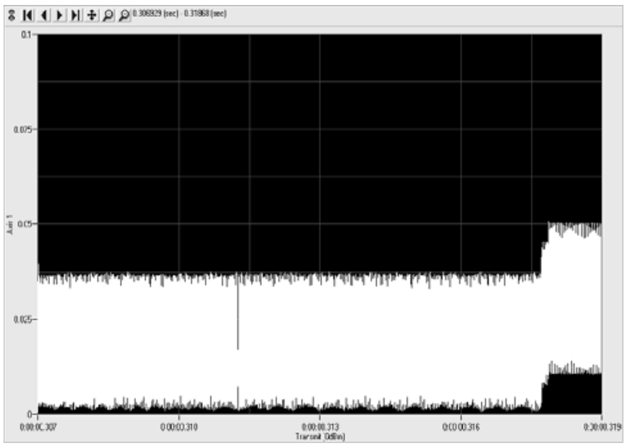
Figure 3: Transmit at 0dBm current vs Time
Vertical scale: 0mA – 100mA
Horizontal scale: 307mS – 319mS

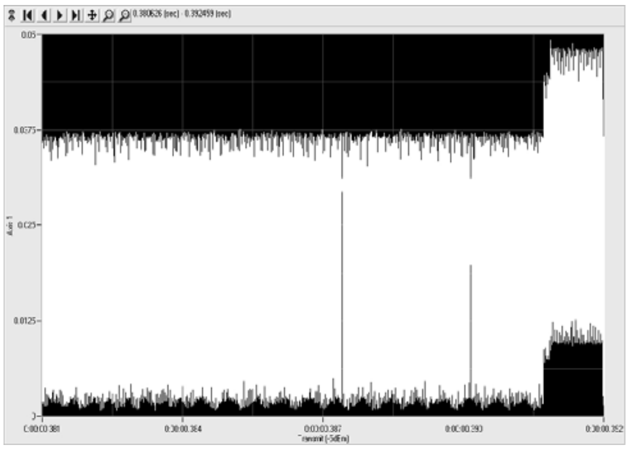
Figure 4: Transmit at -5dBm current vs Time
Vertical scale: 0mA – 50mA
Horizontal scale: 381mS – 392mS

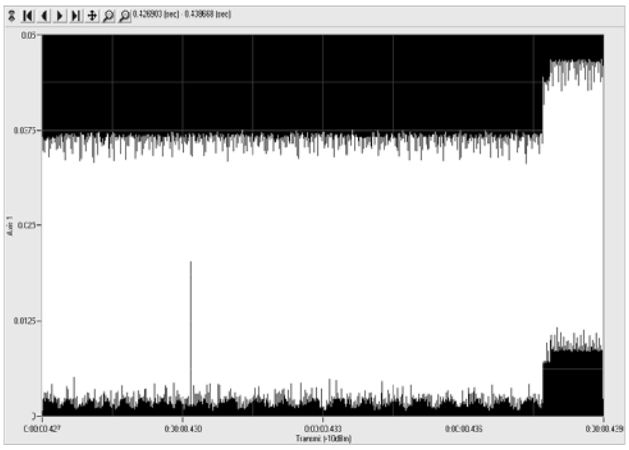
Figure 5: Transmit at -10dBm current vs Time
Vertical scale: 0mA – 50mA
Horizontal scale: 427mS – 439mS

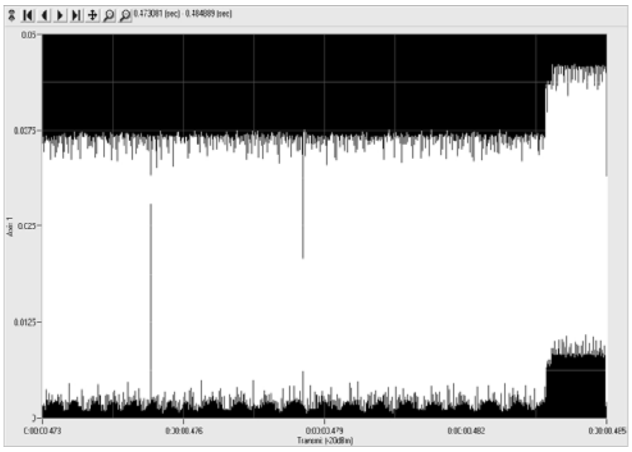
Figure 6: Transmit at -20dBm current vs Time
Vertical scale: 0mA – 50mA
Horizontal scale: 473mS – 85mS

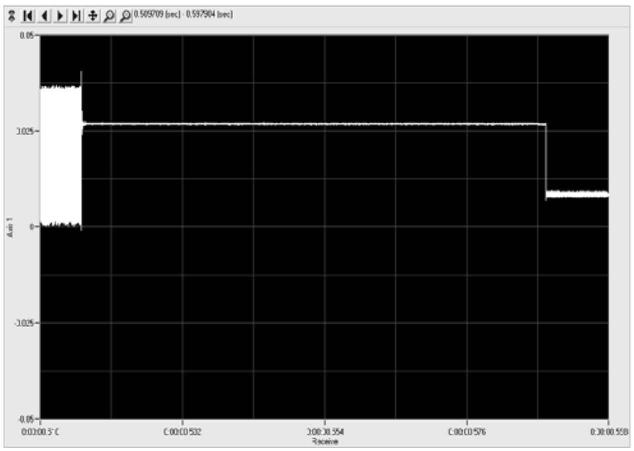
Figure 7: Receive current vs Time
Vertical scale: -50mA – 50mA
Horizontal scale: 510mS – 598mS

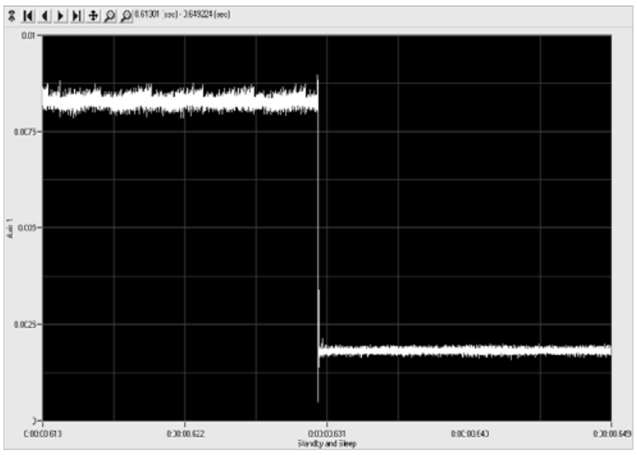
Figure 8: Wireless Standby & CPU Sleep current vs Time
Vertical scale: 0mA – 10mA
Horizontal scale: 613mS – 649mS.

The results of these measurements are summarized in Table 3, and the Dsystem25 parameters derived from these results are presented in Table 4.

Measurement	Current	Duration
Transmitting at 0 dBm	19.46mA	11,727us
Transmitting at -5 dBm	19.28mA	11,790us
Transmitting at -10 dBm	19.16mA	11,721us
Transmitting at -20 dBm	19.10mA	11,688us
Receiver in power-down mode	8.32mA	
Receiver in idle mode	26.5mA	
Receiving	10.2mA	10,622us
Read EEPROM (1 Byte)	8.32mA	
Write EEPROM (1 Byte)	9.85mA	4.2ms
CPU in sleep/power-down	1.82mA	
CPU in active mode	8.32mA	
Write Flash (256 Bytes)	10.19mA	4022us
Erase Flash (256 Bytes)	10.16mA	4021us

Table 3: Dsystem25 current measurement results

The measurements were repeated three times for one node, and then repeated on two additional nodes, resulting in 5 different data sets. Analysis of these results provides a 95% confidence limit of ± 10.5% of the measured current figures.

Parameter	**Value [mA]**
E_{perm}	0.058 uAs/byte
E_{rxbit}	0.484 uAs/bit
E_{txbit}	0.100 uAs/bit

Table 4: Dsystem25 energy parameters

Based on the current measurements for each operation shown, the time and average current for the operation was calculated.

The parameter values are derived by calculating the energy consumed by the node for each operation (power * time), and dividing this by the number of bytes to derive the per-byte figures used in the benchmark equation.

5 Results

Using the power benchmark Equation (6), the parameters from Tables 3 and 4, and representative values for parameters S, h, k, and ρ we derive the following minimum power benchmarks for software updates.

A. MICA2 node benchmark result

h = 40 bits
k = 232 bits (29 bytes)
$E_{benchmark} = 37960$ uA-seconds

B. DSystem25 node benchmark result

h = 40 bits
k = 224 bits (28 bytes)
$E_{benchmark} = 41622$ uA-seconds

The following parameter values were used to represent a typical software update:

S = 8192 (a significant, 8KByte software update)
ρ = 0.000010 (1 bit per 100,000)

These results can be interpreted as follows: on the MICA2 node, the minimum charge required to receive and store an 8KByte software upgrade is 0.037 Coulombs. On the DSystem25 node, the minimum charge required is 0.042 Coulombs.

Note that the intention is not to provide a comparison of hardware platforms, but to provide a benchmark against which software providing software upgrade functionality can be objectively assessed (including both system and 'software-upgrade-application' software). To compare hardware platforms the voltage would need to be taken into account to measure electrical energy in Joules.

6 Future Work

This work presented in this paper represents the completion of this stage of the research. In the next stage, a new model will be built that accounts for the energy consumption associated with distributing the software throughout the network. This will include factors to account for the energy used during advertisement, data transfer, and lost-packet re-requests. The same methodology will be applied to derive a minimal energy equation. It is planned to complete this future stage by taking power measurements for a number of different software update algorithms to compare them with the 'ideal' minimum charge consumption.

The energy models and minimum-energy benchmark methodology presented here could also be used as the basis for developing a minimum energy benchmark for other tasks that run on wireless sensor networks: for example routing and data collection. Future research will involve using the methodology shown here to develop these benchmarks.

7 Conclusions

A model is presented for estimating the minimum charge required to perform a software update on a wireless sensor node. This model is used to produce a minimum energy benchmark equation.

Measurement results from real nodes are presented, and are used with this equation to produce a quantitative and objective benchmark against which the energy effectiveness of software updates mechanisms can be compared on any node (and figures shown for two hardware platforms: the MICA2 and DSystem25 nodes).

The novel methodology described here allows the calculation of an absolute energy utilization reference point against which real protocols can be measured. This provides an absolute measure, or benchmark, as a goal for energy reduction. It is likely that no real node can meet the ideal minimum energy consumption derived by the model, but it provides a target against which node hardware and software can be designed and evaluated. It also provides an objective target against which real implementations can be compared.

The methodology presented in the work is also applicable in developing minimum-energy benchmarks for other tasks that run on a wireless sensor network. For example, a calculation of the minimum energy required to distribute connectivity information would provide the basis for a minim-energy benchmark for connectivity-based routing schemes.

Acknowledgments

We would like to acknowledge the assistance of Philip Angove and Ivan Ferrante in the Tyndall National Institute in matters relating to the DSystem25 hardware, and funding from Science Foundation Ireland under grant NAP18.

References

[1] S. Brown, C. Sreenan. "A New Model for Updating Software in Wireless Sensor Networks", IEEE Networks, Nov/Dec 2006, pp.42-47,(2006).

[2] CrossBow Products. "MICA2 Wireless Measurement System Datasheet", Document Part Number: 6020-0042-08 Rev A, Crossbow Technology Inc,(2007).

[3] M. Galos, D. Navarro, F. Mieyeville, I. O'Connor. "Energy-aware software updates in heterogeneous Wireless Sensor Networks," in Proc. IEEE 9th International New Circuits and Systems Conference (NEWCAS), pp.333-336,(2011).

[4] C. Haas and J. Wilke. "Evaluating the energy-efficiency of key exchange protocols in wireless sensor networks". In Proc. of the 7th ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks(PM2HW2N'12),pp.133-140, (2012).

[5] A. Hagedorn, D. Starobinski, A. Trachtenberg. "Rateless Deluge: Over-the-Air Programming of Wireless Sensor Networks Using Random Linear Codes," in Proc. International Conference on Information Processing in Sensor Networks (IPSN '08), pp.457-466,(IEEE,2008).

[6] J. Hui, D. Culler. "The Dynamic Behaviour of a Data Dissemination Protocol for Network Programming at Scale", in Proc. 2nd Intl. Conference on Embedded Networked Sensor Systems, pp. 81-94,(ACM,2004).

[7] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, K.S.J. Pister. "System architecture directions for networked sensors", in Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 93-104,(ACM,2000).

[8] S. Jayapal, R. Huang, S. Ramachandran, R. Bhutada, Y. Manoli. "Optimization of electronic power consumption in wireless sensor nodes", in Proc. of the 8th Euromicro Conference on Digital System Design, pp. 165-169, (IEEE,2005).

[9] J. Jeong, S. Kim, A. Broad. "Network Reprogramming", TinyOS 1.x Documentation, Aug 12, (2003). (available at http://www.tinyos.net/tinyos-1.x/doc/Xnp.pdf)

[10] J. Koshy and R. Pandy. "Remote Incremental Linking for Energy-Efficient Reprogramming of Sensor Networks", in Proc. of the Second European Workshop on Wireless Sensor Networks, pp.354-365,(IEEE,2005).

[11] T. Liu, and M. Martonosi. "Impala: A Middleware System for Managing Autonomic, Parallel Sensor Systems", in Proc. of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Progamming (PPoPP’03), pp.107-118,(2003).

[12] T. Liu, C.S. Sadler, P. Zhang, M. Martonosi. "Implementing Software on Resource-Constrained Mobile Sensors: Experiences with Impala and ZebraNet". in Proc. of the 2nd Intl. Conference on Mobile Systems, Applications and Services (MobiSys’04), pp.256-269,(ACM,2004).

[13] E. Modiano. "An adaptive algorithm for optimizing the packet size used in wireless ARQ protocols", Wireless Networks 5, pp.279-286, (Kluwer,1999).

[14] B. O'Flynn, S. Bellis, K. Delaney, J. Barton, S.C. O'Mathuna, A.M. Barroso, J. Benson, U. Roedig, C. Sreenan. "The development of a novel minaturized modular platform for wireless sensor networks", Fourth International Symposium on Information Processing in Sensor Networks (IPSN 2005), pp.370-375, (2005).

[15] Pittsburgh. Proc. of the Distributed Sensor Nets Workshop, Pittsburgh, PA, Dept. of Compsci. CMU, (1978).

[16] N. Reijers, K. Langendoen. "Efficient Code Distribution in Wireless Sensor Networks", in Proc. of 2^{nd} ACM Intl. Workshop on Wireless Sensor Networks and Applications(WSNA’03), pp.60-67,(2003).

[17] K. Romer, F. Mattern. "The design space of wireless sensor networks", IEEE Wireless Communications, vol 11, iss. 6, pp.54-61,(2004).

[18] C. Schurgers, V. Tsiatsis, S. Ganeriwal, M. Srivastava. "Optimizing Sensor Networks in the Energy-Latency-Density Design Space", IEEE Trans. Mobile Computing, vol. 1, no. 1, pp.70-80,(2002).

[19] V. Shnayder, M. Hempstead, B. Chen, G.W. Allen, M. Welsh. "Simulating the power consumption of large-scale sensor network applications", in Proc. of the 2nd international Conference on Embedded Networked Sensor Systems (SenSys '04), pp.188-200,(ACM,2004).

[20] S. Singh, C.S. Raghavendra. "PAMAS: Power aware multi-access protocol with signalling for ad hoc networks," ACM Computer Communication Review, vol. 28, no. 3, pp.5-26,(1998).

[21] J.A Stankovic, T.E. Abdelzaher, Chenyang Lu, L. Sha, J.C. Hou. "Real-time communication and coordination in embedded sensor networks", in Proc. of the IEEE, Volume: 91, Issue: 7, pp.1002-1022,(2003).

[22] M. Steine, Cuong V. Ngo, R. S. Oliver, M. Geilen, T. Basten, G. Fohler, J-D. Decotignie. "Proactive re-configuration of wireless sensor networks", in Proc. of the 14th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems (MSWiM '11),(ACM,2011).

[23] L. Wang. "MNP: Multihop Network Reprogramming Service for Sensor Networks", Proc. 2nd Intl. Conference on Embedded Networked Sensor Systems, pp.285-286,(ACM,2004).