This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSEN.2015.2388857, IEEE Sensors Journal

Measuring and Modelling the Energy Cost of Reconfiguration in Sensor Networks

Gowri Sankar Ramachandran, Wilfried Daniels, Nelson Matthys, Christophe Huygens, Sam Michiels, Wouter Joosen, James Meneghello, Kevin Lee, Eduardo Cañete, Manuel Diaz and Danny Hughes.

***Abstract*—As Wireless Sensor Networks (WSN) must operate for long periods on a limited power budget, estimating the energy cost of software operations is critical. Contemporary reconfiguration approaches for WSN allow for software evolution at various granularities; from re-flashing of a complete software image, through replacement of complete applications, to the reconfiguration of individual software components. This paper contributes a generic model for measuring and modelling the energy cost of reconfiguration in WSN. We validate that this model is accurate in the face of different hardware platforms, software stacks and software encapsulation approaches. We have embedded this model in the LooCI middleware, resulting in the first energy aware reconfigurable component model for sensor networks. We evaluate our approach using two real-world WSN applications and demonstrate that our model predicts the energy cost of reconfiguration with 93% accuracy. Using this model we demonstrate that selecting the most appropriate software modularisation approach is key to minimising energy consumption.**

***Index Terms*—Wireless Sensor Networks, Software Reconfiguration, Energy Modelling.**

I. Introduction

Software reconfiguration is a critical issue for Wireless Sensor Networks (WSN) due to two factors. Firstly, the cost and complexity of deploying a WSN necessitates that infrastructure can support multiple applications throughout its lifetime [1]. Secondly, the resource constraints of WSN necessitate optimisation of software configurations to suit changing environmental conditions. As WSN are often deployed at scale in inaccessible or dangerous locations such as flood plains [2], all reconfiguration must be enacted remotely. Furthermore, as WSN must execute for long periods on a limited power budget it is important to accurately predict the energy cost of reconfiguration actions in order to plan system configuration.

Research from the field of WSN has resulted in a variety of software evolution approaches, which may be categorised by their granularity:

Monolithic reconfiguration approaches allow for reconfiguration through replacement of the entire software image running on each mote, including all operating system (OS) and application functionality. This approach is exemplified by TinyOS [3].

Application-based approaches, such as Contiki [4] and Squawk [5] separate OS functionality from application functionality, allowing for the replacement of complete application images.

Component-based approaches such as OpenCOM [6] and LooCI [7] allow for the replacement of individual components within an application at runtime.

Contemporary software evolution techniques for WSN have two key shortcomings. First, they do not quantify the energy cost of reconfiguration, which makes it difficult for application developers to reason over reconfiguration options. Second, the relative costs of each reconfiguration approach have yet to be fully evaluated in realistic WSN scenarios.

In contrast to our prior work [8], this paper provides a complete treatment of the energy modelling problem. Specifically, we contribute a generic model and methodology for calculating the energy cost

Gowri Sankar Ramachandran, Wilfried Daniels, Nelson Matthys, Christophe Huygens, Sam Michiels, Wouter Joosen and Danny Hughes, are with iMinds-DistriNet, KU Leuven, 3001 Leuven, Belgium.
E-mail: gowrisankar.ramachandran@cs.kuleuven.be
James Meneghello is with Murdoch University, Perth, Australia.
Eduardo Cañete and Manuel Diaz are with University of Malaga, Malaga, Spain.
Kevin Lee is with School of Science and Technology, Nottingham Trent University, Nottingham, UK.

Copyright (c) 2013 IEEE. Personal use of this material is permitted. However, permission to use this material for any other purposes must be obtained from the IEEE by sending a request to pubs-permissions@ieee.org.

1530-437X (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSEN.2015.2388857, IEEE Sensors Journal

of software evolution in WSN. We evaluate our approach on three different WSN hardware platforms (SPOT [9], Raven [10] and Zigduino [11]), using different programming languages (C and Java ME), different encapsulation approaches (ELF and Compact ELF) and operating systems (Contiki [4], SQUAWK [5] and TinyOS [3]). We find that our energy model has an average accuracy of 93%.

The second unique contribution of this paper is an exploration of the efficiency of each class of software evolution approach (monolithic, application-based and component-based) for two real-world applications: waste bin tampering and smart parking. Our results indicate that different approaches offer distinct energy trade-offs during configuration, thus reconfiguration of software must be carefully considered by the software developer.

The remainder of this paper is structured as follows: Section II provides background on reconfiguration and software evolution in WSN. Section III describes the case-study scenarios. Section IV describes a generic model and associated methodology for calculating the energy cost of reconfiguration. Section V models the LooCI middleware running on various hardware/software stacks. Section VI evaluates the accuracy of this energy model in two real-world scenarios. Section VII discusses related research. Finally, Section VIII concludes and discusses directions for future work.

II. Reconfiguration in WSN

In this section, we provide key examples of software evolution in WSN and from these distill requirements for energy-aware reconfiguration.

The SICS factory surveillance system described in [12] used a 25-mote WSN to monitor conditions in a factory complex. Motes were powered by limited batteries and all application functionality was implemented as a set of Contiki modules [4]. In this paper, the authors highlight the need for both autonomic *adaptation* and heteronomous *evolution* of deployed application functionality to meet changing application requirements. The GridStix flood monitoring and warning system [2] was deployed on the river Ribble in northern England and the river Dee in northern Wales. The system used a *heterogeneous* architecture composed of embedded Linux boards powered by batteries and solar panels to monitor conditions on a 1KM stretch of river. All application functionality was implemented using the OpenCOM component model. In [2] Grace et al. demonstrate that component-based *adapation* can be used to optimise application behaviour to meet changing environmental conditions. The Cambridge badger monitoring experiment [13] used a WSN to monitor the behaviour of badgers in a nature reserve. The application used a heterogeneous architecture with RFID tags deployed on the collars of badgers, static RFID detection stations and battery-powered motes that monitored the local microclimate. Application functionality was implemented as Contiki application modules [4]. After deployment, software was subject to heteronomous *evolution* based upon input from domain experts and to accommodate changes in the hardware platform.

Considering the examples of software reconfiguration discussed above, it can be seen that reconfiguration serves two general purposes: (i.) to heteronomously evolve application functionality to meet changing requirements and (ii.) to automatically optimise application functionality to suit changing environmental conditions. Successfully enacting energy-aware reconfiguration in resource constrained mote environments gives rise to three requirements:

1) *Accurate Energy Models* enable informed reasoning over software adaptations or evolution, it is critical to consider the energy cost of reconfiguration actions.
2) *Support for Heterogeneity* is critical as the applications discussed above run on different hardware, operating systems and languages.
3) *Guidance on selecting reconfiguration approaches* is required to assist developers in choosing an encapsulation and modularisation approach.

III. Case Study Applications

We introduce two case-study applications: smart parking and waste bin tampering. These case-study applications were provided by OneAccess, an international company with facilities located in Belgium. OneAccess have developed a common hardware/software network architecture that is used to support both case studies. This architecture features a common networking approach and four tiers of functionality:

1) **Network:** IPv6 is supported end-to-end using RPL [14], which updates the routing tables

1530-437X (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSEN.2015.2388857, IEEE Sensors Journal

of each mote in the router tier and connects these to the sensor tier. All motes use industry standard IEEE 802.15.4 radios.

2) **Sensor tier:** based on a custom embedded mote platform that offers a 16MHz Atmel MCU, 16KB RAM and 128KB flash memory. *Sensor motes* monitor their local environment, execute simple analysis algorithms on sensor data and send the results to the *router tier*. The sensor tier does not participate in multi-hop routing. Instead all sensor nodes are leaves which connect to a node in the router tier. Sensor motes are powered by two AA batteries, which must last for over 5 years without replacement to ensure economic viability.
3) **Router tier:** based on the same embedded mote platform as the sensor tier, *router motes* offer multi-hop routing functionality, relaying data between the sensor tier and the gateway tier . The routers form a tree topology with a gateway as the root and the sensors as the leaves. The routers are installed in fixed locations and are powered from the electricity grid.
4) **Gateway tier:** based on an Alix embedded PC platform, which offers a 500MHz CPU, 256MB RAM, 802.15.4 and 802.11 networking. The gateway runs embedded Linux and Java SE. The gateway is powered directly from the electricity grid and bridges the WSN running RPL over 802.15.4 and community wireless networks running standard 802.11 WiFi.
5) **Back-end tier:** is comprised of powerful servers on high-speed connections, which gather data from all sensors and expose processed results to users via a web interface.

In order to minimise costs, OneAccess uses common routing, gateway and back-end infrastructure to support both sensing applications. The *sensor mote* tier is then extended as needed by specific sensing applications.

A. Smart Parking Application

The smart parking application makes parking more efficient by using motes to monitor free parking spaces and communicating this information to drivers via their smart phone.

Sensor motes are embedded in a durable case known as a 'speeddisk', which is commonly deployed on roads to slow down traffic. The speed disk is attached to the concrete at street level and is capable of withstanding the pressure of cars driving over it. Each mote is equipped with a magnetometer, which measures the local magnetic field on three axes and an Infra-Red (IR) distance sensor. The magnetometer is polled once per second to detect whether a car has arrived or left. The battery level of the mote is polled once per minute. When a car arrives or leaves, an update message containing a boolean value representing parking space availability and the current battery level is forwarded to the routing tier and from there to the gateway. The back end system then publishes updated parking space availability to the web user interface.

B. Waste Bin Tampering Application

The waste bin tampering application monitors when public waste bins are opened and closed in order to detect tampering. The bins should only be opened by staff, whose schedule is known. Where the opening time of the bin differs from the staff schedule, tampering is identified and a cleaning crew is dispatched to fix the problem.

For this application, sensor motes are extended with a magnet-activated reed switch which detects when the bin is opened or closed. When a hardware interrupt is generated by the reed switch, the time that the bin was opened is stored in flash memory. This log of open/close times is sent to the gateway every 24 hours.

As with the smart parking system, the waste bin tampering system has been deployed for testing at small scale in the city of Ghent. OneAccess anticipate that further reconfiguration will be necessary to optimize system behaviour based upon observed environmental conditions.

IV. Energy Aware Reconfiguration Model

The following sections introduce our generic reconfiguration model and describes how this model is parameterized for a specific platform.

A. Generic Modeling Approach

The energy cost of a reconfiguration can be broken down into the cost of discovering the current configuration using *introspection*, the cost of

1530-437X (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSEN.2015.2388857, IEEE Sensors Journal

deploying new functionality and the cost of *configuring* the deployed functionality. The total energy consumption of any reconfiguration can therefore be defined as:

$$E_R = E_D + E_I + E_C. \quad (1)$$

Where E_D is the energy consumption of the component deployment, E_I is the energy consumption of introspection calls and E_C is the energy consumption of configuration calls. It is intuitive that the energy consumed during the deployment of new software functionality will have a positive linear relationship to the size of the software that is being deployed. E_D may thus be calculated using the following linear regression equation:

$$E_D = \sum_{i=1}^{n}((CS_i \times \beta_1) + \beta_0). \quad (2)$$

Where β_0 is the minimum cost of a deployment operation, β_1 defines the relationship between component size and additional energy consumption, CS_i refers to the size of the i^{th} component and n denotes the number of component deployments.

The energy consumption of introspection (E_I) and control (E_C) operations are simply the sum of the cost of each operation, given by:

$$E_I = \sum_{i=1}^{n} E_i. \quad (3)$$

$$E_C = \sum_{i=1}^{n} E_i. \quad (4)$$

Where n refers to the number of introspection or control commands and E_i denotes the energy consumption of i^{th} introspection or control commands. This generic model must be calibrated for each hardware/software stack that is modelled.

B. Energy Calibration Methodology

We use the energy measurement methodology presented in [8]. This method requires a digital output pin on the test mote platform and the ability to instrument the code under test to signal when measurement should start and stop. A Digital Storage Oscilloscope (DSO) is used to estimate the power consumption of the mote under test. A digital IO pin is toggled on the mote platform to indicate the starting and the ending point of the API call. The energy consumed by the software API call is derived using the Ohm's law. We refer the reader to [8] for a detailed description of the measurement methodology.

V. Modeling The LooCI Middleware on Heterogeneous Motes

We now calibrate the generic energy model and energy measurement methodology to create a specific energy model for the LooCI middleware running on the Zigduino [11], Raven [10] and SPOT [9] motes.

A. The LooCI Middleware

We model the reconfiguration API of the Loosely-coupled Component Infrastructure (LooCI) [7], a component-based middleware for sensor networks that was developed by our group. LooCI provides a good test platform as it runs on heterogeneous hardware/software stacks and thus affords the opportunity to demonstrate that our energy modeling approach is *generic* (i.e. not tied to a specific hardware platform, operating system or programming language). The complete LooCI reconfiguration API is shown in Listing 1. Full details are provided in [7].

Listing 1: The core LooCI API

```
CompID    deploy(ComponentFile, NodeID)
Boolean   removeComponent(CompID, NodeID)

Boolean   deactivate(CompID, NodeID)
Boolean   activate(CompoID, NodeID)
Boolean   wireLocal(EventType, SourceCompID,
              DestCompID, NodeID)
Boolean   wireFrom(EventType, SrcCompID, SrcNodeID,
              DestCompID, DestNodeID)
Boolean   wireTo(EventType, SrcCompID, SrcNodeID,
              DestNodeID)

CompID[]  getComponentIDs(NodeID)
String    getComponentType(NodeID, CompID)
State     getComponentState(NodeID, CompID)
Event[]   getInterfaces(NodeID, CompID)
Event[]   getReceptacles(NodeID, CompID)
```

LooCI currently supports three *underlying platforms*: Contiki [4], Squawk [5] and OSGi [15]. LooCI layers standard support for remote component deployment, introspection and configuration on

1530-437X (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSEN.2015.2388857, IEEE Sensors Journal

top of these heterogeneous systems, exposing the common API shown above. *Deployment commands* allow for the insertion and removal of software components. *Introspection commands* allow for the discovery of what components are present on a node, their interfaces, state and bindings. Finally, *Configuration commands* allow for the activation, deactivation and binding of components. All commands may be enacted remotely at runtime.

B. Mote Platforms

We model the energy cost of the LooCI reconfiguration API on three platforms:

The **Zigduino** offers a 16MHz Atmega128RFA1 MCU with built-in 802.15.4 radio, 16KB RAM and 128KB flash [11]. The Zigduino runs either TinyOS [3] or Contiki OS [4] and the LooCI middleware [7]. All software is implemented in C.

The **Raven** offers a 20MHz ATmega1284PV MCU, 16KB RAM, 128KB flash and AT86RF230 IEEE 802.15.4 radio [10]. The Raven motes run an identical software stack to the Zigdunio.

The **SPOT** offers a 180MHz ARM920T MCU, 512KB RAM, 4MB flash and CC2420 IEEE 802.15.4 radio [9]. The SPOT motes run the Squawk OS and JVM [5] and the LooCI middleware [7]. All software is implemented in Java.

As can be seen from the specifications above, these motes are heterogeneous in terms of hardware resources, operating systems and languages and are thus an ideal selection of sensing technologies on which to demonstrate the *generic* nature of our approach.

C. Energy Models

As described in Section IV-A, any reconfiguration action in a reconfigurable component model such as LooCI is composed of a set of *deployment* and *configuration* operations, while *introspection* commands allow component configurations to be validated before and after reconfiguration.

As expected, *deployment* operations consume orders of magnitude more energy than *introspection* or *configuration* commands due to the transmission of component functionality. This is shown in Figure 1(a), 1(c) and 1(e), wherein the red cross shows the average energy cost of the component deployment and the red line indicates the 95% confidence interval. The maximum size of an over-the-air deployable component is 3.2KB on the Zigduino and Raven, while the maximum component size on the SPOT is 40KB.

Figure 1(b), 1(d) and 1(f) shows the energy cost of non-deployment remote API calls for Zigduino, Raven and SPOT hardware platforms respectively. We have represented the 95% confidence interval with blue bars for each API call. The average energy cost of *non-deployment* API calls follows a similar trend on both of our experimental platforms, except that the energy cost of removing a component consumes a large amount of energy on the SPOT due to the energy used when accessing flash memory. The average energy consumption is 3mJ for the Raven, 13mJ for the Zigduino and 225mJ for the SPOT. The wide variation in energy consumption clearly demonstrates the need for per-platform calibration of the energy model. Particularly when one considers that the specifications of the Raven and Zigduino are very close, yet the API calls to the Zigduino consume more than four times as much power.

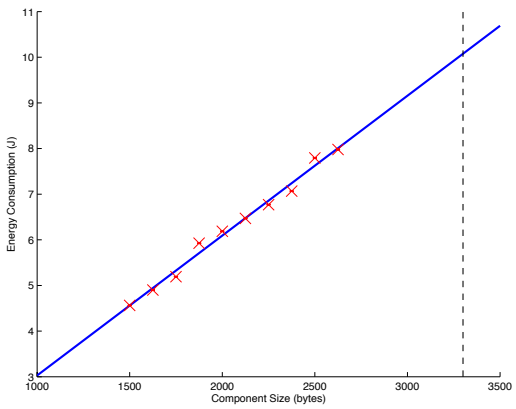
Figure 1(a), 1(c) and 1(e) shows that, there is a positive relationship between the energy cost of deployment and component size. A linear regression equation was computed from our sample data and is shown by the blue line in Figure 1(a), 1(c) and 1(e). Our linear energy model for the component deployment captures the relationship between the energy cost of the deployment and the component size. This model can therefore be used to obtain the energy cost of any component deployment for Zigduino, Raven and SPOT hardware platforms.

VI. Implementation and Evaluation

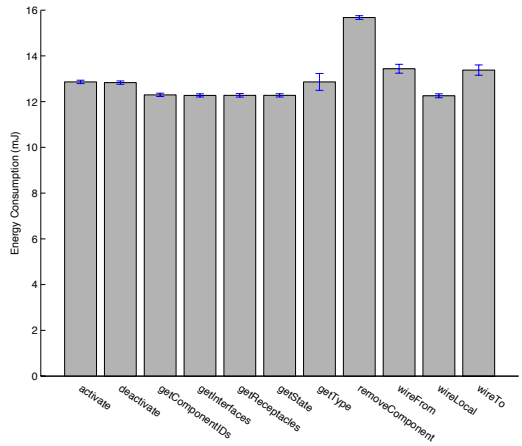
All reconfiguration in LooCI is enacted via a manager component which runs on the network gateway. The manager accepts and executes simple scripts of reconfiguration API calls. We extended this script interpreter to provide energy cost estimates for all reconfiguration scripts using the model presented in the previous section. This provides developers with a simple mechanism to assess the energy costs of reconfiguration actions before they are enacted. In the following section we explore the accuracy

1530-437X (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

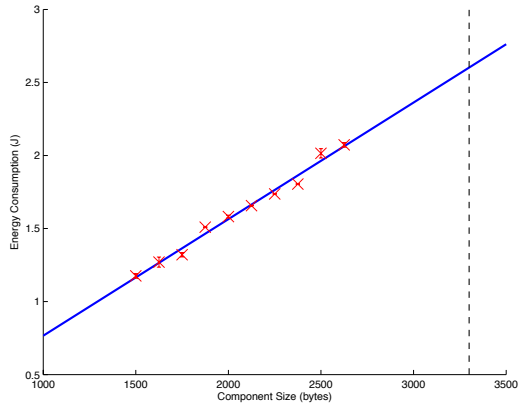
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSEN.2015.2388857, IEEE Sensors Journal



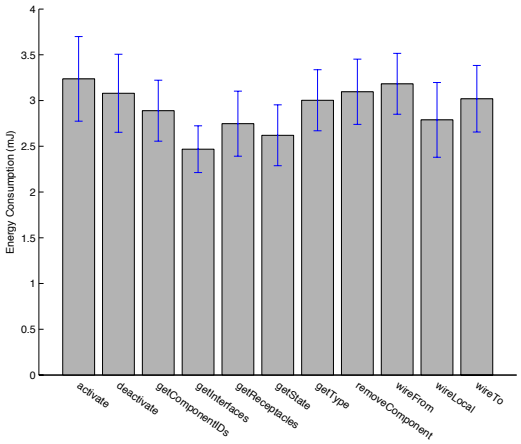
(a) Zigduino component deployment



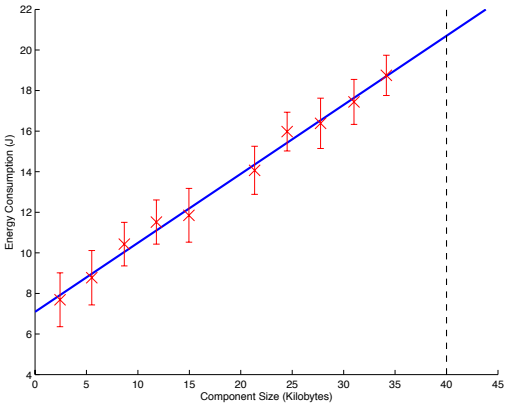
(b) Zigduino reconfiguration/introspection



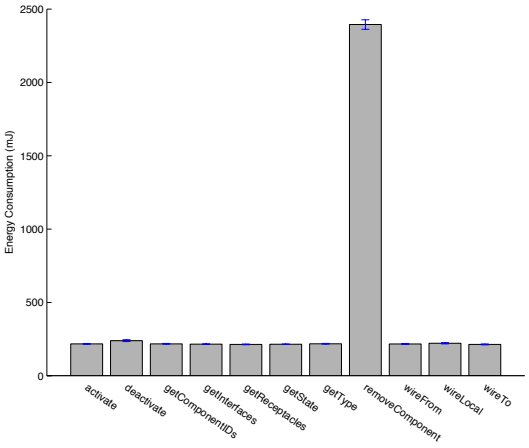
(c) Raven component deployment



(d) Raven reconfiguration/introspection



(e) SPOT component deployment



(f) SPOT reconfiguration/introspection

Fig. 1: Energy cost of LooCI API calls.

1530-437X (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSEN.2015.2388857, IEEE Sensors Journal

of the model in predicting the energy cost of example reconfiguration scripts for our case-study applications. All experiments report the energy cost of reconfiguration for a single mote in a one-hop network.

A. Case-study Evaluation

To evaluate our energy model, four versions of the smart parking and waste bin tampering applications were developed. The first version was developed as a single TinyOS image. The second version was developed as a single Contiki application. The third was developed as a single SQUAWK application. Finally, the fourth was developed as a composition of LooCI components. A series of reconfiguration scripts were then run to discover the accuracy of the energy model and the impact of encapsulation and modularization on artifact size and energy consumption. All experiments were conducted 10 times.

Experiment 1 - effect of encapsulation on size: Table I and Table II show the impact of different encapsulation technologies on code-size. The standard ELF encapsulation format incurs an average size overhead of 304% compared to functional code, while our custom Micro-ELF format reduces this overhead to 175%. The SQUAWK suite format is significantly larger than either ELF variant. These results show the critical importance of appropriate encapsulation formats in reducing energy consumption.

Experiment 2 - effect of modularisation on size: As can be seen from Figure 2, monolithic application implementations are more than twice as large as component based applications. This is because all necessary OS-level functionality must be included with application code. Application-based implementations have the smallest footprint for initial configuration, but necessitate larger code updates during reconfiguration, where component-based reconfiguration reduces update size by 29%. Application-based modularization is thus most efficient for static scenarios, while component-based modularization will be more efficient for dynamic scenarios.

Experiment 3 - energy cost of configuration: This experiment evaluates the costs of remotely configuring a blank mote with a new application. In the case of the single-unit application, no configuration is necessary and therefore all energy cost is due to the deployment of functionality. In the case of the component-based application, components had to be deployed, configured and activated. The results of our energy evaluation are presented in Tables III and IV, which shows that (i.) our energy model is on average 97.83% accurate on the Raven and 89.23% accurate on the SPOT and (ii.) initial configuration is least efficient using the monolithic TinyOS approach, which consumes over 9.3 Joules, while the component-based approach consumes over 3.4 Joules and an application-based configuration consumes the least energy at 1.3 Joules. The greater error in predicting the energy consumption of the SPOT is attributed to the non-deterministic behaviour of the Squawk JVM. The poor performance of monolithic configuration is primarily due to the necessity of including OS functionality with application code updates.

Experiment 4 - energy cost of reconfiguration: The second experiment tackles a problem which emerged for OneAccess during small-scale testing of their system. The magnetometer sensor was proving hard to calibrate and so the application was reconfigured to also use the IR sensor to detect when cars arrived or left. In the case of a component based application, this change necessitates only the deployment, wiring and activation of a new IR sensor component. In the case of application based development, it requires redeployment of the complete application image. In the case of monolithic development this requires monolithic reflashing. The results of our energy evaluation are presented in Tables III and IV, which show that: (i.) our energy model is 96.33% accurate on the Raven and 93.08% accurate on the SPOT and (ii.) component-based reconfiguration results in energy savings of between 18.98% and 32.52% compared to a single application implementation. The monolithic approach of TinyOS is very inefficient for reconfiguration, consuming 6.4 times as much energy as the component-based approach.

In summary, it can be seen that our generic energy modelling approach is accurate for application-based and component-based applications running on heterogeneous OS and hardware platforms. Furthermore, our analysis reveals that while component-based reconfiguration has a significantly higher

1530-437X (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSEN.2015.2388857, IEEE Sensors Journal

	Contiki			SQUAWK
	Data Size (bytes)	Standard ELF (bytes)	Micro ELF (bytes)	SPOT Suite (bytes)
Smart Parking				
Manager	192	1052	692	2404
Sensor	354	1352	934	2256
Filter	240	1124	752	2117
Battery	330	1328	910	2229
Total	1116	4856	3288	9006
Waste Bin Tampering				
Manager	464	1536	1074	2443
Sensor	326	1312	894	2061
Battery	330	1328	910	2228
Total	1120	4176	2878	6732

TABLE I: Size of component-based application composition for LooCI.

	Contiki			SQUAWK
	Data Size (bytes)	Standard ELF (bytes)	Compact ELF (bytes)	Suite Size (bytes)
Smart Parking	484	1686	1176	6289
Waste Bin Tampering	514	1716	1196	6048

TABLE II: Size of the single application implementation for Contiki.

	Raven			SPOT		
	Model (mJ)	Benchmark (mJ)	Accuracy (%)	Model (mJ)	Benchmark (mJ)	Accuracy (%)
Smart Parking	3885	3887	99.94	32876	29851	90.79
Bin Tampering	3387	3484	97.21	24594	20612	83.81
Smart Parking Reconfiguration	1089	1124	96.33	7780	7260	93.08
		Average	97.83		**Average**	89.23

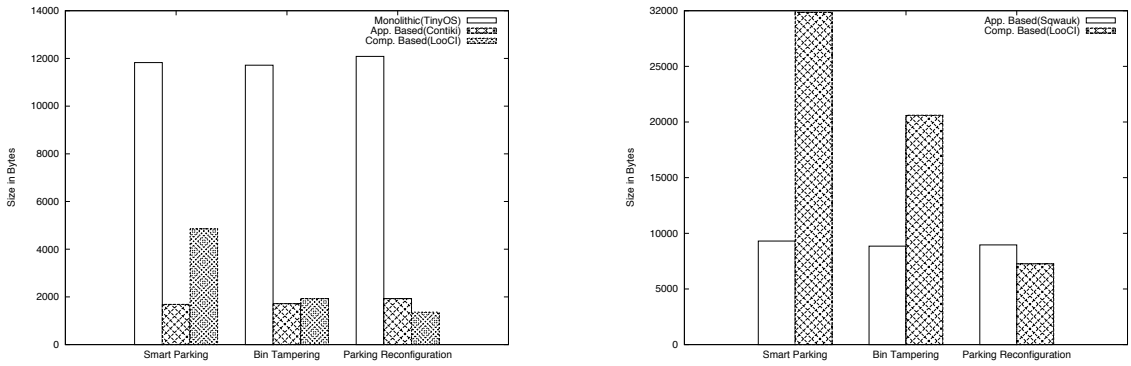
TABLE III: Accuracy of energy model in predicting the cost of over-the-air configuration and reconfiguration.

	Effect of modularisation (on energy)		
	Comp-based (mJ)	App-based (mJ)	Monolithic (TinyOS) (mJ)
Raven			
Smart Parking	3887	1311	9408
Bin Tampering	3484	1336	9321
Smart Parking Reconfiguration	1124	1594	9611
SPOT			
Smart Parking	29851	9302	-
Bin Tampering	20612	8847	-
Smart Parking Reconfiguration	7260	8961	-

TABLE IV: Impact of component-based reconfiguration on application size and energy consumption.

1530-437X (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSEN.2015.2388857, IEEE Sensors Journal



(a) Artefact size on Raven and Zigduino

(b) Artefact size on SPOT

Fig. 2: Artefact Size on Zigduino, Raven and SPOT hardware platforms.

initial cost than application-based reconfiguration, incremental software updates are significantly more efficient, which over the life-time of the sensor network is likely to compensate for the initial configuration overhead. In contrast, monolithic development methodologies are inefficient for both Over-The-Air (OTA) configuration and reconfiguration.

VII. Related Work

A. Analytical Methods

Analytical models provide designers with the possibility of evaluating the lifetime of their WSN applications in a fast and platform-independent way. In [16], the authors propose a probabilistic lifetime energy model based upon the relationship between the lifetime of a single mote and the whole sensor network. A different approach is presented in [17] where a state-based battery model is proposed to accurately estimate the battery life. A key problem of these methods is that they only consider the energy consumption related to packet transmission and reception. These approaches therefore provide poor estimates when the energy consumption of other devices such as a CPU or sensors is significant. For example, in the Great Duck Island experiment [18] it was noted that EEPROM use on the Mica 2 mote during reprogramming consumed four times as much energy as transmitting using the radio. A similar result is evident in our SPOT energy measurements, which show that radio use is not the only factor that should be considered when estimating energy usage. In our view it is therefore imperative to specifically calibrate any energy measurement approach for the specific hardware/software platform offered by each mote.

B. Experimental methods

A more commonly-used methodology for measuring energy consumption is based on the use of an oscilloscope, an operational circuit connected to the target node and a program executing on a PC to analyse the data obtained from the oscilloscope [19], [20]. We build upon this energy monitoring approach, while adding software instrumentation to automate the testing process and providing a model that accurately predicts the energy consumption of software operations.

Landsiedel et al. [21] provide a tool called AEON to model and predict the energy consumption of a sensor network application. This tool measures the energy cost of individual hardware component on a sensor node and use that cost to predict the lifetime of a TinyOS [22] sensor network application. This approach has two key limitations. First, the model requires a white-box understanding of the devices present on each mote. Second, the approach is tied to TinyOS [22]. In contrast, our approach may be applied to any software/hardware stack by following a simple and uniform energy calibration process.

Sankar et al. [23] compare the performance of WSN OSs. The software evolution model was found to

1530-437X (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JSEN.2015.2388857, IEEE Sensors Journal

have a direct relationship to reconfiguration efficiency. During the execution phase, the monolithic configuration performs better than the application or component-based reconfiguration. However, during reconfiguration, more fine-grained approaches perform significantly better. Our work validates and complements the results presented in [23] by providing a more in-depth analysis of reconfiguration energy costs.

VIII. Conclusions and Future Work

This paper addressed the problem of predicting the energy cost of reconfiguration operations for WSN middleware through two contributions: (i.) a generic energy model and (ii.) the first energy-aware runtime reconfigurable component model. Evaluation shows that our energy model accurately predicts the energy consumed by reconfiguration actions on three heterogeneous mote platforms: the SPOT [9], the AVR Raven [10] and the Zigduino [11]. We also provide guidance on how to select modularisation and encapsulation approaches based on application characteristics. In sum, addressing the requirements highlighted in Section II.

Our future work will extend the work performed in this paper on system level energy modelling to consider the runtime energy consumption of applications.

References

[1] C. Huygens, D. Hughes, B. Lagaisse, and W. Joosen, "Streamlining development for networked embedded systems using multiple paradigms," *IEEE Software*, 2010.

[2] D. Hughes, P. Greenwood, G. Blair, G. Coulson, P. Grace, F. Pappenberger, P. Smith, and K. Beven, "An experiment with reflective middleware to support grid-based flood monitoring," *Concurr. Comput. : Pract. Exper.*, 2008.

[3] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," *SIGPLAN Not.*, 2000.

[4] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki-a lightweight and flexible operating system for tiny networked sensors," in *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, 2004.

[5] D. Simon, C. Cifuentes, D. Cleal, J. Daniels, and D. White, "Java on the bare metal of wireless sensor devices: the squawk java virtual machine," in *Proceedings of the 2nd international conference on Virtual execution environments*, 2006.

[6] G. Coulson, G. Blair, P. Grace, F. Taiani, A. Joolia, K. Lee, J. Ueyama, and T. Sivaharan, "A generic component model for building systems software," *ACM Trans. Comput. Syst.*, 2008.

[7] D. Hughes, K. Thoelen, J. Maerien, N. Matthys, J. D. Cid, W. Horre, C. Huygens, S. Michiels, and W. Joosen, "Looci: a loosely-coupled component infrastructure for networked embedded systems," in *Proceedings of the 11th IEEE International Symposium on Network Computing and Applications*. IEEE, 2012.

[8] D. Hughes, E. Caete, W. Daniels, R. Gowri Sankar, J. Meneghello, N. Matthys, J. Maerien, S. Michiels, C. Huygens, W. Joosen, M. Wijnants, W. Lamotte, E. Hulsmans, B. Lannoo, and I. Moerman, "Energy aware software evolution for wireless sensor networks," in *World of Wireless, Mobile and Multimedia Networks, 2013 IEEE 14th International Symposium and Workshops on a*, 2013.

[9] Oracle, "Sun SPOT - Theory of Operation," 2012. [Online]. Available: http://www.sunspotworld.com/docs/Yellow/SunSPOT-TheoryOfOperation.pdf

[10] "AVR RZ Raven Datesheet," 2012. [Online]. Available: http://www.atmel.com/Images/doc8117.pdf

[11] "Zigduino Manual," 2013. [Online]. Available: http://wiki.logos-electro.com/zigduino-r2-manual

[12] N. Finne, J. Eriksson, A. Dunkels, and T. Voigt, "Experiences from two sensor network deployments self-monitoring and self-configuration keys to success," in *Wired/Wireless Internet Communications*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, vol. 5031.

[13] V. Dyo, S. A. Ellwood, D. W. Macdonald, A. Markham, C. Mascolo, B. Pásztor, S. Scellato, N. Trigoni, R. Wohlers, and K. Yousef, "Evolution and sustainability of a wildlife monitoring sensor network," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, 2010.

[14] "RPL: IPv6 routing protocol for low-power and lossy networks," 2012. [Online]. Available: http://tools.ietf.org/pdf/rfc6550.pdf

[15] "OSGi Core Release 5," 2012. [Online]. Available: http://www.osgi.org/download/r5/osgi.core-5.0.0.pdf

[16] K. Sha and W. Shi, "Modeling the lifetime of wireless sensor networks," *Sensor Letters*, 2005.

[17] J. Rahmé and K. Al Agha, "A state-based battery model for nodes' lifetime estimation in wireless sensor networks," in *Proceedings of the tenth ACM international symposium on Mobile ad hoc networking and computing*, 2009.

[18] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, 2002.

[19] C.-C. Chang, D. J. Nagel, and S. Muftic, "Assessment of energy consumption in wireless sensor networks : A case study for security algorithms," in *Proceedings of IEEE International Workshop on Wireless and Sensor Networks Security*. IEEE, 2007.

[20] K. Shinghal, A. Noor, N. Srivastava, and R. Singh, "Power measurements of wireless sensor networks node," *IJCES*, 2011.

[21] O. Landsiedel, K. Wehrle, and S. Gotz, "Accurate prediction of power consumption in sensor networks," in *Embedded Networked Sensors, The Second IEEE Workshop on*, may 2005.

[22] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," in *Proceedings of the 9th international conference on Architectural support for programming languages and operating systems*, 2000.

[23] G. S. Ramachandran, S. Michiels, W. Joosen, D. Hughes, and B. Porter, "Analysis of sensor network operating system performance throughout the software life cycle," in *Network Computing and Applications (NCA), 12th IEEE International Symposium on*, 2013.

1530-437X (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.