Journal of Systems Architecture 59 (2013) 626–642

Contents lists available at SciVerse ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

PASES: An energy-aware design space exploration framework for wireless sensor networks

Ivan Minakov, Roberto Passerone *

Dipartimento di Ingegneria e Scienza dell'Informazione, University of Trento, via Sommarive 5, Povo di Trento, TN, Italy

ARTICLE INFO

Article history:
Received 31 December 2012
Available online 6 June 2013

Keywords:
Embedded systems
Design space exploration
Wireless sensor networks
Platform based design
Energy aware

ABSTRACT

Energy consumption is one of the most constraining requirements for the development and implementation of wireless sensor networks. Many design aspects affect energy consumption, ranging from the hardware components, operations of the sensors, the communication protocols, the application algorithms, and the application duty cycle. A full design space exploration solution is therefore required to estimate the contribution to energy consumption of all of these factors, and significantly decrease the effort and time spent to choose the right architecture that fits best to a particular application. In this paper we present a *flexible and extensible* simulation and design space exploration framework called "PASES" for accurate power consumption analysis of wireless sensor networks. PASES performs both performance and energy analysis, including the application, the communication and the platform layers, providing an extensible and customizable environment. The framework assists the designers in the selection of an optimal hardware solution and software implementation for the specific project of interest ranging from standalone to large scale networked systems. Experimental and simulation results demonstrate the framework accuracy and utility.

1. Introduction

The design of networked embedded systems and wireless sensor networks (WSN), in particular, faces unique challenges which are specific to their application domains. These are rendered difficult to approach due to the tight system requirements. Among the existing issues, power consumption and energy efficiency are considered to be one of the most constraining factors that limit functionality and mass production of WSN systems. Sensor nodes are commonly resource-constrained, battery-powered embedded devices. Replacing the batteries is, however, at best inconvenient, and sometimes downright impossible after the network deployment. Hence WSN nodes must utilize power efficient hardware and software technologies to increase their lifetime.

The study of power optimization can be approached at several levels. From a technology and hardware point of view, techniques can be applied to dynamically scale the voltage of the circuits or regulate the frequency of operation according to the computation needs. While these techniques may provide substantial savings, the choice of the application algorithms, operating systems, protocol implementations, scheduling policies and program style may have a considerable impact on energy consumption. Additionally, collaborative network behavior, topology, MAC and routing protocol functionalities often have dominant influence on the network lifetime. Evaluating the effects of different design choices in each of the mentioned domains is therefore a critical step in the design of the applications and in the sensor network development process. Reliable and accurate energy evaluation can be achieved only by capturing all relevant low-level details and operating states of the studied hardware platform along with proper modeling of application functionality, communication behavior, network and sensing environments.

The most common approaches for the evaluation of WSN systems include modeling (simulation) and prototyping. Prototyping is a widely used method which provides an efficient and highly valuable evaluation of the tested systems. However, despite its obvious benefits, prototyping has a number of crucial disadvantages that have to do with the high cost and time required to build, run and debug experimental systems. Among the modeling approaches, simulation is currently the most widely used technique to explore and evaluate the design of WSN applications. A vast number of tools for WSN simulation have been introduced to perform power consumption evaluation from the network point of view [1–3]. In these environments, the system is modeled as a set of communicating concurrent processes, where each process represents the activity of a single node. Existing simulators [1,4,2,5–14] provide various degrees of support for the analysis of communication, application logic and energy consumption. Many tools are focused either primarily on the network or software

* Corresponding author. Tel.: +39 0461 283971.
E-mail address: roberto.passerone@unitn.it (R. Passerone).

1383-7621/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.sysarc.2013.05.020

simulation, without proper care for the underlying hardware platform details. These tools perform energy consumption estimation based on simple assumptions and energy models that account only for basic operating states (e.g., On/Off) of the Radio-Frequency (RF) transceiver and the processor. Other tools exploit a cycle-accurate simulation strategy to examine operations with fine grained details but for a specific hardware architecture only, without the possibility to extend them. While these tools provide very accurate power estimation, their strong focus on a particular hardware architecture limits their applicability in design space exploration. Only quite few of the available tools provide enough flexibility to evaluate energy usage for a wide range of hardware WSN platforms organized into a network.

In this paper we present a *flexible* and *extensible* SystemC-based framework called Power Aware Simulator for Embedded Systems (PASES) that combines an event-driven simulation engine with a hardware, application and network models composer. The main motivation behind PASES is to enable accurate energy estimation for a wide range of WSN hardware platforms and fill up the gap between purely network oriented WSN tools and architecture specific simulation environments. This is especially important when evaluating the trade off between computation and communication. This tool allows a designer to construct models of any target HW/SW platform, organize them into a heterogeneous network and estimate the power consumption of every individual node and get an energy-timing load profile for each HW component of interest. It also supports the design, optimization and analysis of collaborative network behavior, communication protocols, application algorithms and enables one to study their effects on the network throughput, delivery latency, energy efficiency and other metrics. PASES is based on a methodology which supports the Platform Based Design (PBD) paradigm [15,16], providing power analysis for various heterogeneous sensor platforms organized in a network by defining separate abstraction layers for application, communication, hardware, power supply modules and sensing environment (stimulus).

This paper is structured as follows. We first review the state of the art on WSN simulation tools in Section 2. In Section 3 we discuss the underlying methodology of our framework. Section 4 provides the implementation details of the presented framework and describes the ways in which a system model can be constructed. Finally, we demonstrate a case study application in Section 5 and evaluate the framework.

2. Related work

Design space exploration of wireless sensor networks is generally accomplished through modeling and simulation tools. We roughly classify existing simulation tools into *general purpose* (generic) network simulators, *network oriented* and node oriented frameworks. Generic network simulators are designed to model and evaluate conventional networks, such as wired LAN TCP/IP or IEEE 802.11 WiFi systems. A number of frameworks, such as the open source NS2 [1], NS3 [4] and OMNet++ [2] and the commercial OPNet [5] and QualNet [6], have become popular in this domain. The other two classes of tools are more specific to our target application domain, and provide different degrees of design space exploration capabilities. Network oriented simulation tools focus on the network aspects of the WSN systems and typically offer highly realistic and accurate models of the communication infrastructure. Node oriented simulators, referred in literature either as sensor node simulators or emulators, target internal operations of sensor nodes while still providing simple lightweight communication models. Our aim is to provide a balanced framework, that can be customized to trade off the accuracy of the model with the performance of the simulation. For this reason, PASES supports both networked node simulation and platform modeling, including models of the energy source to support lifetime estimation. In the rest of the section we briefly highlight the recent state-of-the-art with respect to WSN exploration tools. A full survey and comparison of the PASES framework with several of these tools is under preparation and subject of a separate publication.*Network Oriented WSN Simulators*

A number of tools are based and extend the OMNet++ framework to provide specific analysis capabilities for WSNs. Castalia [7] and MiXiM [8] are excellent WSN simulators which provide a first-order analysis of algorithms and protocols before their actual implementation on a specific node platform. These tools provide detailed radio and communication modeling, including a highly customizable and accurate radio PHY model with dynamic calculation of radio signal strength, interference and noise ratio depending on the modulation type. Additionally, Castalia includes customizable models of the most popular MAC and routing protocols for WSN applications. However, these tools provide power aware modeling and power consumption reporting for the radio component only, and do not include power models for hardware peripherals typical of a node platform (Timers, ADC, etc.). Additionally, they do not provide any battery model and hence they do not support lifetime estimation. Another disadvantage of Castalia and MiXiM is their lack of an efficient post processing tool with GUI support.

Also based on OMNet++, PAWiS [9] is a simulator focused on power analysis for a wide range of WSN applications. It provides facilities to simulate a whole sensor network along with accurate modeling of the internal activity for every single node in the network. Its architecture provides an extensible and flexible simulation infrastructure, where users can customize and add new simulation modules and components. Hardware components represent hardware models of the sensor node, including the CPU, timers, ADC and the radio transceiver. The tasks that simulate the work of the dedicated hardware are used to report the power consumption to the power supply module. The biggest disadvantage of PAWiS is its poor documentation and lack of clear design examples, which makes it quite difficult to use and get started with. Additionally, PAWiS is no longer supported, the latter release was issued in 2009.

WSNet [10] is a standalone simulation tool written in C++ for Linux. WSNet offers a wide range of radio medium models from basic ideal physical layers to complex and accurate ones that include fading, frequency correlation and different antenna radiation patterns. The simulated nodes in WSNet are built as a composition of blocks which represent either a hardware component or a software component of the node. Additionally, WSNet provides an option to simulate dynamic physical phenomena (e.g., a fire) and physical measures (e.g., temperature, humidity). This feature allows sensor-driven applications to be simulated and analyzed. The lack of a GUI environment for interpreting the simulation results is a disadvantage of the WSNet tool-kit.

DANSE [11] is a recent standalone Windows simulator with rich GUI support. This tool was originally designed for teaching purposes and hence it is optimized for ease of use. It provides an excellent GUI to configure various model options, run simulation and study results. DANSE offers a relatively simple communication channel model which however captures the most relevant transmission aspects. Besides, this tool-kit provides a set of the most popular MAC protocols models. Of interest, DANSE includes an experimentally based power model for the radio PHY component (CC2420). Despite its rich GUI features and high level of usability, DANSE does not provide any input interface for the user to define new applications, MAC models or specific hardware options, that in fact limits the applicability of this tool.*Sensor Node Simulators*

PowerTOSSIM [12] is a TOSSIM [17] extension that provides power analysis for TinyOS applications. This simulator provides code level simulation of TinyOS applications which can be run on actual sensor network hardware. PowerTOSSIM equips the TOSSIM hardware model with a specific module, PowerState, to keep track of state activities for each simulated hardware component. Additionally, PowerTOSSIM provides CPU profiling support to report the number of cycles for each execution block. By combining the timing information with the energy model of the simulated hardware platform, PowerTOSSIM computes the total energy consumed by the node. However, PowerTOSSIM is included only in the first version of the TinyOS framework, which is not currently supported. Besides, simulation in TOSSIM and PowerTOSSIM are supported only for a single platform (Micaz [18]).

AVRORA [13] is one of the most widely used WSN emulation tool. It exploits cycle accurate instruction simulation to run the application code. AVRORA runs actual application binaries without the need to specially adapt them for simulation. AVRORA includes the AEON evaluating tool [19] to estimate the power consumption of sensor nodes. AEON exploits the cycle accurate execution model of AVRORA for precise timing and energy measurements. However, AVRORA and AEON support solely the AVR MCU cores [20] and do not provide any extensions for other architectures. In addition, AEON supports energy modeling only for the Mica2 platform [18].

The ATEMU [14] tool-kit is an emulator that targets AVR-based WSN platforms (only the MICA2 platform is supported). This tool exploits a cycle-by-cycle strategy to run the binary code of a target application, and provides a set of accurate models of the peripheral components included in the MICA2 platform. For the network simulation, a simple free space propagation model is implemented. ATEMU also provides an efficient GUI, called XATDB, that can be used to debug and observe the execution of the code, supporting breakpoints and other debugging options. This tool provides the highest level of accuracy among the available WSN emulation environments. However, this is achieved at the cost of high processing requirements and poor scalability. Another issue related to the ATEMU tool is its capacity to emulate only the AVR-based nodes, which makes it impossible to study WSN systems based on other platforms.

3. Methodology

The complexity of developing and modeling heterogeneous embedded and wireless sensor systems requires the use of an efficient system level design methodology based on a set of appropriate abstraction levels. The underlying concept of the presented framework is inspired by the Platform Based Design (PBD) paradigm [21,15,16], which was initially proposed for design optimization of complex electronic systems. PBD structures the design process as a sequence of refinement steps that leads from the initial specification toward a final system implementation. This methodology is a "meet in the middle" approach which combines both the top-down and bottom-up design and refinement flows. The main principle of PBD is a clear definition of intermediate abstraction layers (platforms) for each design stage. Each layer in turn is a set of design components (library) whose implementation details are hidden from the higher layers. Thus, the design process on each stage is a mapping between two adjacent layers where design components from the lower layer are chosen in order to meet higher layer requirements.

The design flow supported in our methodology follows a cyclic refinement process according to the PBD concept as depicted in Fig. 1. The top level functional specification defines requirements and sets constraints down to the application and communication spaces. These in turn propagate constrains down toward the hardware resources. The choice of hardware resources defines the node architecture, i.e., an executable platform with a set of interconnected hardware components which meet the application needs. The energy-performance parameters are mapped on the node model obtained in the previous step. The choice of power supply module completes the system modeling and produces a complete sensor node platform instance. Finally, the node model can be simulated and evaluated to check its correctness with respect to the initial requirements.

Applying the PBD approach for power aware modeling of embedded systems, we identify a set of separate levels of abstraction and corresponding specification layers to capture various design aspects and explore various solutions on each separate stage, as shown in Fig. 1. At the highest level, the **Software Level**, we introduce the *Application Layer* (AL) that is used by the end user to define the application functionality. The *Communication Layer* (CL) is a library of communication protocols which can be chosen and tuned to meet network requirements such as throughput, latency, energy efficiency and so on. At the **Architecture Resource Level** (ARL), the *Service Layer* (SL) presents a collection of functional services (API) that can be composed to specify different applications. Their implementation makes use of the *Resource Behavioral Layer*, which defines a set of hardware resources (components) constituting the target architecture, and the *Resource Annotation Layer*, which specifies their energy-performance details. At the **Energy Level**, the *Energy Source Layer* collects energy sources for sensor nodes.

We start the description of our methodology from the **Architecture Resource Level** that offers a set of HW resources to be assembled into a virtual platform model. Each hardware component on this level is represented as an independent HW entity which provides functionality and specific features related to a certain kind of peripheral (e.g., CPU, ADC, Timer, Sensors and Radio). Every HW model on this level is split onto two layers in order to separate the component behavior from its timing-energy details. The *Resource Behavioral Layer* is introduced to represent the HW behavior intrinsic of certain kind of HW peripherals. The models on this layer define the internal HW operations and data processing which occur in the real HW. The power-performance details appear on the *Resource Annotation Layer*, which defines a set of operating states and transition rules for each separate component. Further, each operating state and transition is annotated with numeric energy-performance values related to the target HW. These values are obtained either from the specification or from real measurements. Thus, the mapping of these two layers provides a high level of flexibility in HW design space exploration since, in most cases, it is enough to simply change either transition rules or annotation numbers in order to model HW components from different vendors. The details of the HW models definition and annotation are given in Section 4, where we describe the framework implementation.

The complete HW sensor node architecture is a set of dedicated behavioral components annotated with energy-performance information. In our methodology we distinguish between *mandatory* and *optional* components in the node HW architecture. The set of mandatory components includes the CPU and clock resource modules which always have to be present in a platform model. The CPU module acts as a central component that manages access to other optional peripherals, handles interrupt requests (IRQ) and executes application tasks. The choice of optional components and their features are left to the user to support a broad application space and various HW platforms.

The **Architecture Resource Level** exposes only the available HW functionality and hides the implementation details from the software developers. This leads to the *Service Layer* (SL), which provides an application interface (API) that encompasses all possible

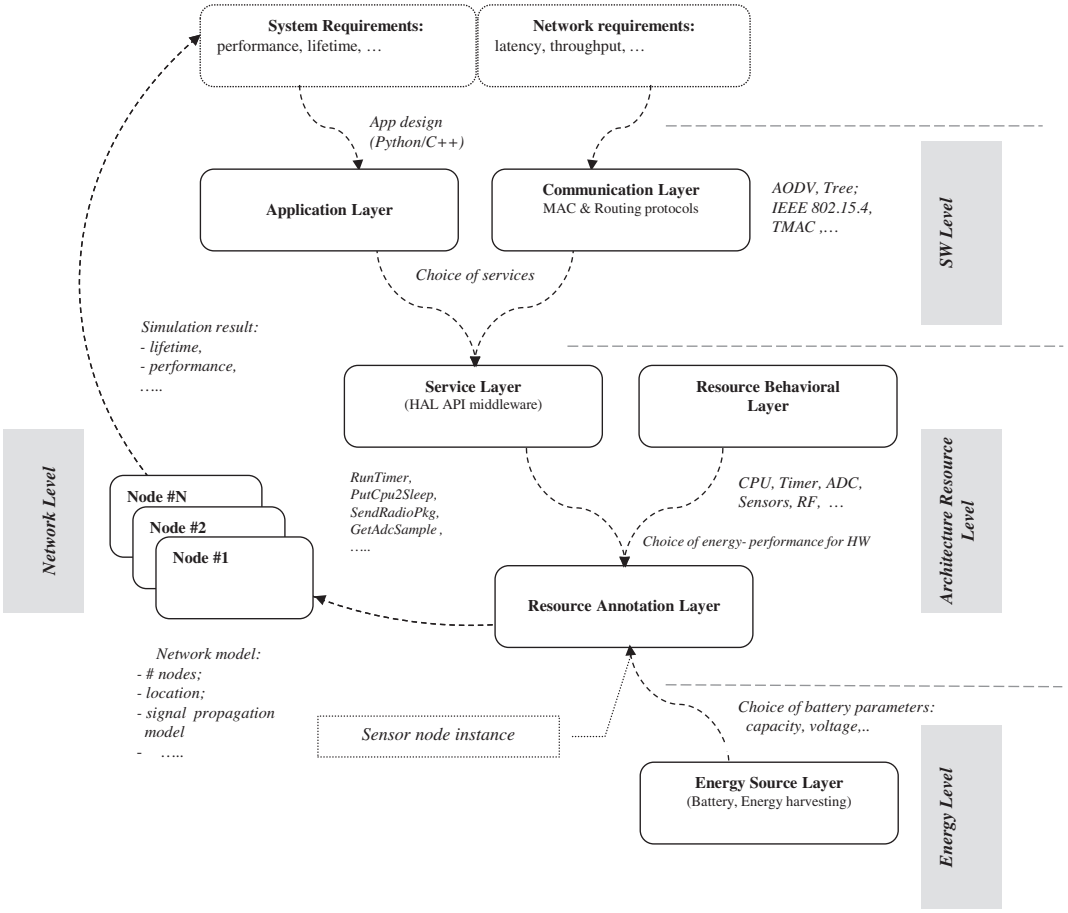


Fig. 1. Design space exploration methodology.

SW services that can be used by an application. The mapping between a dedicated HW model and a set of special services forms a Hardware Abstract Layer API (HAL API) for a certain kind of peripheral component. The HAL API essentially is the middle-ware through which an application interacts with the underlying hardware.

At the above level, the **Software Level**, the *Application Layer* exists to define an application through a set of services provided by the *Service Layer*. This is an open layer, where the application description is left to the user. We imply that every modeled application can be described as a combination of hardware abstract services (HAL API) and user defined algorithms. The application is then executed on behalf of the CPU component which runs the SW tasks sequentially. In our model, we assume that only one single software instruction or user task can be run at a time. The execution flow may be interrupted only if an Interrupt Request (IRQ) event generated by some of the HW component (such as Timer or ADC) occurs. The main execution flow in this case is suspended for the period of time needed to handle the IRQ. Processing is performed inside the user defined interrupt service routine (ISR vector) associated with the IRQ. This realistic execution model closely represents the event scheduling and handling operations which take place in real systems. Thus, it opens the possibility to analyze real embedded SW applications and map them onto our virtual executable platform. The details on this process will be given in the next section.

A library of communication protocols composes the *Communication Layer* (CL). This is an optional layer which includes models of MAC and Routing protocols. We distinguish the communication layer from the application to make it possible to refine and change the communication part independently of the application and vice versa. From the application perspective, this layer provides a set of unified and dedicated services to access the functionality of the chosen protocol. The unified network services provide abstract communication primitives supported by each model on this layer. These include generic operations on network transmission, receiving, retrieving address information and so on. Dedicated services contain specific functions related to each specific protocol.

To model inter-node communication and network cooperative behavior, we introduce the **Network Level** (NL) which represents the wireless communication medium. This level processes signal propagation and handles packet transmissions depending on the relative positions of the nodes, signal strength, receiver sensitivity, data packet length, etc. This level provides a set of low-level link services which are accessible through the Radio Transceiver component.

At the bottom of the abstraction stack, the **Energy Level** introduces the *Energy Source Layer* in order to analyze the power consumption and estimate lifetime of the system model. This layer contains models of various energy sources including models of batteries and super capacitors. Each hardware component consumes a certain amount of power in each particular state, which is accounted for by a power meter and eventually by the battery model. This layer may also include harvesting components or other sources of energy, which opens the possibility to analyze power management techniques.

Finally, the complete platform instance is represented as a vertical composition of configured components sequentially taken and

refined on each level of abstraction described above. It provides an efficient simulation infrastructure which allows one to capture and analyze details on each of the layer including application, hardware, energy and communication. Fig. 2 shows an example of HW platform layers composition and top down mapping related to an ADC peripheral. The arrows represent the refinement process where platform specific options are selected on different layers, e.g., available API, set of operating states, ADC resolution and timing features, current draw depending on supplied voltage, clock frequency, etc.

4. Framework implementation

The PASES framework is designed to assists the development, modeling and optimization of networked embedded systems and of WSNs in particular. It provides precise per-node power and timing analysis, capturing aspects and details related to the simulated hardware platforms at the desired level of granularity. The framework is implemented as a standalone MS Windows application designed in Visual Studio C++. To meet all the requirements described in the previous section, PASES utilizes a component-based architecture consisting of a simulation kernel; a component library (models of HW components); platform and application constructors; a power tracer and power source models; communication stack libraries; a wireless channel model; and a GUI front-end. This section describes each of these components in details. The PASES architecture is presented on Fig. 3.

4.1. Simulation kernel

The core element of PASES is a simulation kernel built on top of the SystemC discrete-event simulation engine. To implement an executable platform model composed of heterogeneous HW components, we adopted a cycle approximate TLM abstraction paradigm [22], which leads to an increased simulation speed and lower design effort with respect to a cycle accurate approach.

The **Architecture Resource Level** introduced in our methodology is implemented as a collection (library) of independent models of HW components. Each component in this library is a conventional SystemC module (sc_module) which represents the behavior of the dedicated hardware (such as Timer, ADC, Radio, etc.). These models do not contain any information related to the timing or energy consumption features of the real hardware and they are only able to perform abstract actions associated with the provided services. PASES provides a basic hardware model class (hw_module) which must be extended by each behavioral model in order to be integrated with the simulation infrastructure. This parent component provides a set of basic communication and functional primitives for the simulation environment including system and component tracers, IRQ controller and energy-performance models. By using the hw_module class as a parent we hide the structure of the underlying layers and provide plain methods to control the component power states.

Each HW model must be associated with the services which that model provides to the application. Components are responsible for handling command messages issued from the *Service Layer*. These commands are requests to perform certain actions, such as sending a packet, run a timer, and so on. The hw_module class provides pure virtual method (cmdHandler) to be implemented as a message handler in a derived class.

Our framework provides a set of pre-designed customizable models for most HW components typically present in a WSN node architecture. These include models of CPUs, Radios, Timers, ADCs, Sensors, USART (SPI, I2C), Flash memory, LEDs and many others. However, the architecture of our framework allows the user to design and easily include new custom components into the PASES component library.

4.2. HW component annotation

The implementation of the *Resource Annotation Layer* for each HW model is performed by defining a component specific Finite State Machine (FSM) and by annotating its states and transitions with energy-performance values related to a real world HW. The set of specific FSMs for one kind of components with associated values forms a library from which one or more components can be chosen to build a specific HW platform. Each library provides a unified export–import interface to exchange information with the behavioral model. By doing so, different platforms can be exchanged by simply substituting elements from the library. This approach effectively separates the behavioral side of the modeled hardware from its energy-performance aspects.

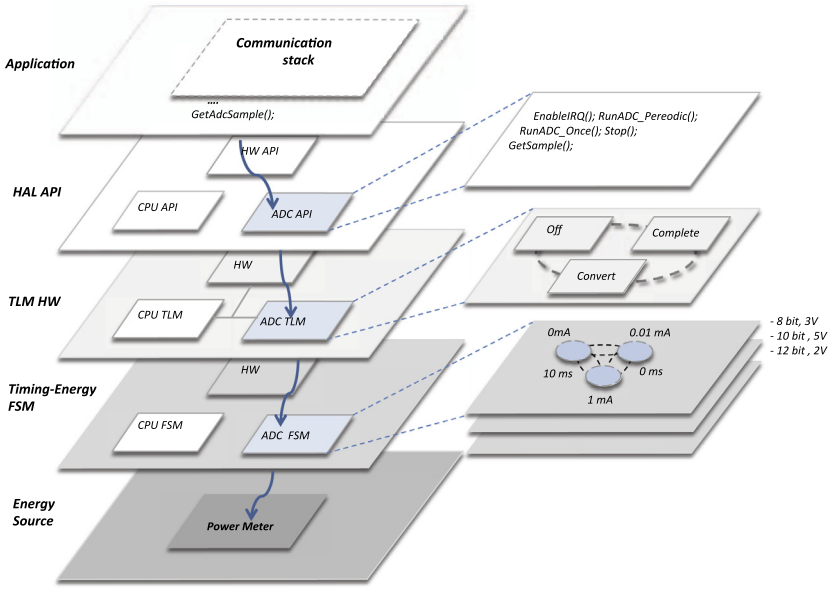


Fig. 2. Platform abstraction layers.

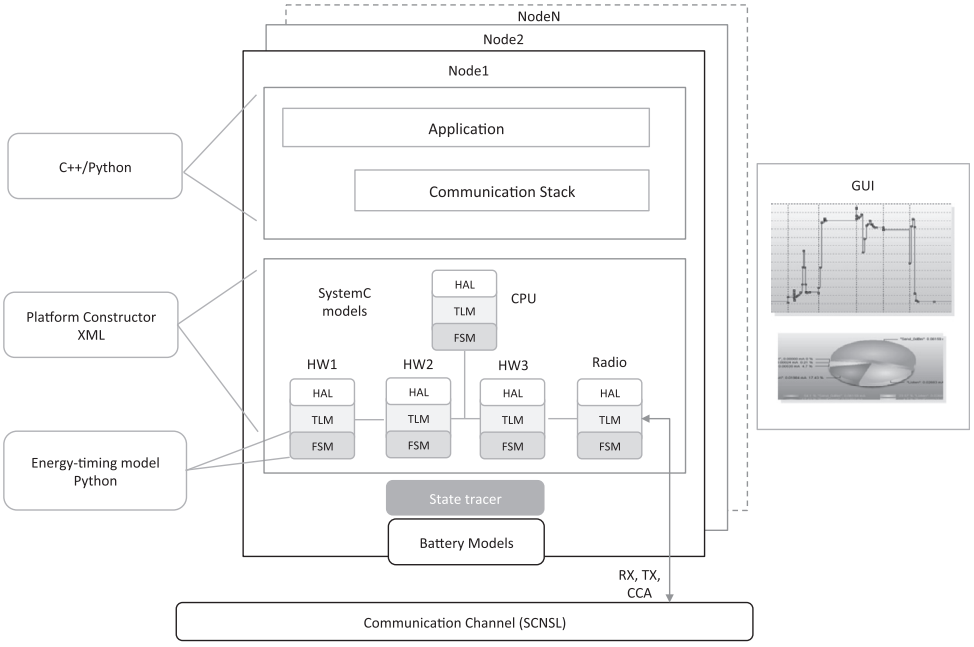


Fig. 3. PASES architecture.

The implementation of the Resource Annotation library resides inside external Python files which in turn provide an easy way to change the model parameters without the need of a compilation step. The Python built-in objects (lists) and special classes are used to hold and export values to the PASES simulation core. The component FSM definition consists of the following basic elements:

- a table of operating states;
- a component state transition table;
- a table with performance information;
- an export–import interface.

The table of operating states contains the set of possible power states for each HW component. Additionally, each HW component might be supplied with a customizable state-transition table which describes the component behavior as a series of reactions to requests generated from upper layers. The component state-transition table allows the user to specify a hierarchical FSM with macro and micro states. Along with a set of operating states (macro states), this table may include a set of intermediate states that are needed to describe the low level edge effects of power switches. We refer to them as *micro states*, which are defined by transition vectors. These vectors are used to perform a more accurate state transition between two macro-operating states. Technically, a vector is a set (Python list) of power states with current consumption and state duration information for each micro state. An example of a vector and its power trace are shown in Fig. 4. The set of intermediate states can be quite complex for some HW components and operations, such as CPUs and radios, which may significantly increase the simulation time. However, the vectors can be easily skipped if there is no need for high simulation precision or if HW state transitions happen instantly, so that the user can trade off simulation performance for accuracy.

A single HW model may hold a number of different FSM definitions corresponding to the different operating conditions, such as different clock frequencies for the CPU core, different output power for the radio and so on. The choice of these conditions may have considerable impact on the system power consumption. PASES provides the ability to change these parameters at run time, thus opening the possibility to analyze different power management techniques.

4.3. Platform construction

The actual choice of a node platform is left to the user to be defined using a description written in XML. The platform instance is created according to this XML specification where a set of peripheral components with associated annotation values are described. Here the user defines the names of the components, their composition and hierarchy, parameters and links to the platform specific FSMs definitions. The component parameters may include an initial CPU clock frequency, ADC resolution modes, etc. Furthermore, in the node architecture we distinguish between on-chip peripheral modules (e.g., Timers, ADC, USART) and external components (e.g., Radio, LEDs, sensors, etc.), which are not part of a single MCU chip. PASES includes an option to setup an external MCU interface to simulate the communication of the MCU with the off-chip components.

4.4. Sensors and environment model

The PASES HW component library contains customizable models for digital and analog sensor devices. For an accurate timing and power analysis, we implemented models of serial interface controllers to connect digital sensors and other peripherals (including the Radio) to the CPU module. Currently we support simple functional models of IIC and SPI controllers. The ADC channel is used to connect analog sensors to the MCU. The component interfaces and interconnections are described in the platform XML specification.

Aside from application algorithms, results of sensing operations may have considerable impact on an application functionality and consequently on the system power consumption. Depending on the application purpose, the deviation of sensed phenomena may trigger the execution of energy consuming tasks such as network transmission, increase rate of the application duty cycle, etc. PASES provides an efficient interface to supply sensor devices with individual dynamic environment models. The dynamic variation of

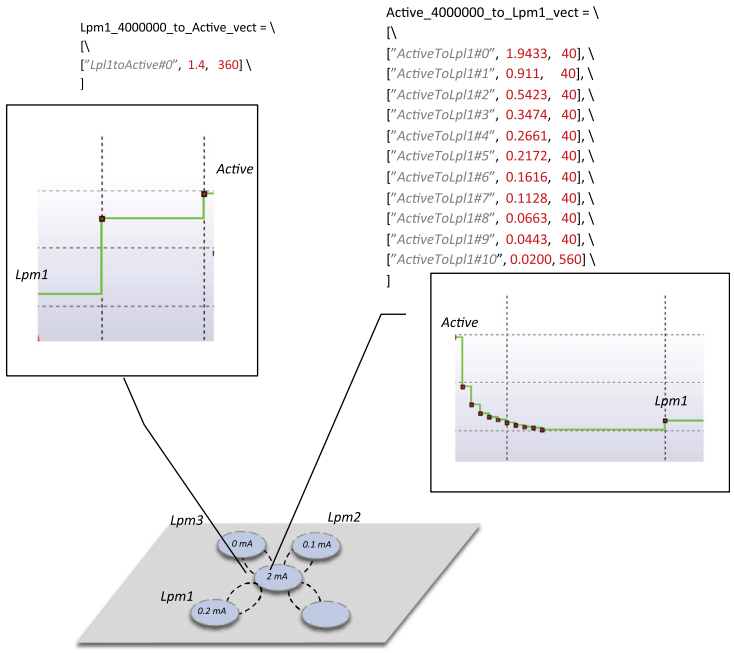


Fig. 4. An example of state transition vectors and associated power traces.

sensed quantities is implemented inside external Python scripts that provide a flexible way to create complex environmental scenarios. The user can utilize any convenient method to define variations of sensed phenomena over simulation time. It may include lookup tables, equations, combinations thereof, and reading from external log files.

4.5. Application construction

The application is implemented on top of the *Service Layer* (SL), which plays the role of an interface layer between the HW platform and the application layers. The application determines the actual functionality of the modeled system. It manages the workload of the HW peripheral modules, switches the components operating states and manages all the data transfers. The application is represented as a sequential execution of SL services and user-defined functions. The available API services provide a comprehensive set of functional primitives to operate with the underlying hardware and build up any model of a target application functionality, such as custom task scheduling, network routing, power state control, run-time clock and voltage scaling. In addition, our framework provides an easy way to extend the predefined set of services with user functions for each specific HW component of interest.

PASES supports application models written either in C/C++ or in Python. Python offers more flexibility over C for a fast application prototyping, while still providing an efficient execution performance.

4.5.1. Embedded profiler

The PASES framework includes a profiling facility that enables accurate timing analysis and simulation of real embedded software code written in C. A software profiler performs timing estimation of the target application by annotating the original C code with detailed cycle accurate timing information. Fig. 5 shows a diagram of the PASES profiler toolchain for both C and Python applications.

Along with timing analysis, PASES provides automated instrumentation to map native source code on a virtual HW platform instance. The output of this process is PASES executable code, where platform dependent operations are replaced with HAL API. The

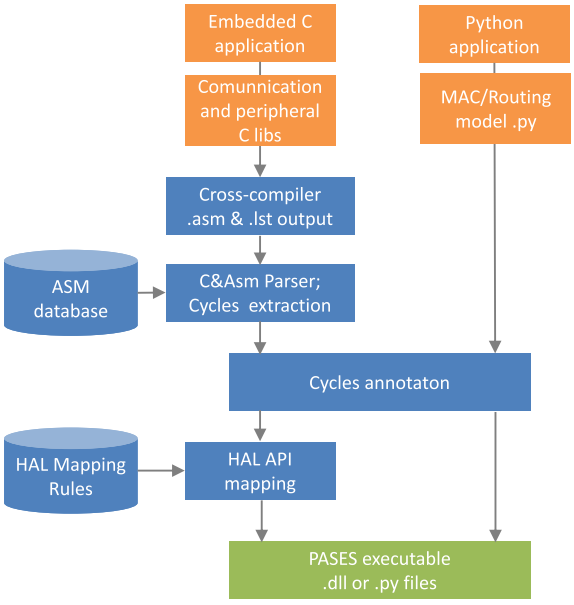


Fig. 5. PASES application toolchain.

mapped application is then executed within the simulator instruction by instruction, advancing the simulation clock according to the annotated time.

Timing analysis is a multi-step process which involves a number of operations. First, we determine the number of cycles needed to execute each individual source line on a target platform. We derive this value from the debug information (listing) files generated by the cross compiler, which allows us to perform an automated correlation between C and assembler instructions. Using this mapping and a look-up table holding the number of cycles required for each individual CPU instruction, the execution time of each source line can be computed. Then, every line in the C file is annotated with the corresponding number of cycles. The functional mapping takes place on the next step, where platform specific operations

such as registers writings are changed to the PASES *Service Layer* API methods. The correspondence between the API and the platform specific code is defined by means of a customizable table which declares replacement rules. While the mapping utility replaces platform specific blocks in the original code, the rest of the application functionality, such as tasks scheduling, network packets processing, and computational algorithms, remain unchanged. However, the complexity of providing reliable mapping of simulation primitives on an arbitrary HW/SW application forced us to limit the number of supported architectures for automatic (semi-automatic) application mapping to only MSP430 [23] and AVR128L [20]. Moreover, the original C application for the supported platforms has to satisfy some specific code conventions in order to enable automatic mapping. Nevertheless, mapping can be done manually by the user in more difficult cases.

The entire tool-chain including the grammar parser is implemented in Python. The C-parser is the main part of our analysis tool. It contains customizable grammar definitions to identify C statements in the source code and defines places for timing annotation depending on the type of statement. For instance, timing information is placed inside the loops, before conditions and function calls, etc. Currently, PASES provides analysis for the code generated by the most popular cross-compilers including msp_gcc [23] for the MSP430 architecture and avr_gcc [20] for the AVR architecture.

4.6. Communication stack

The communication stack in PASES consists of three sub-layers: PHY, MAC/Routing and Application. The lower sub-layer in the stack is a model of the physical network interface (PHY) which is the functional part of the transceiver model. It provides basic access primitives to the network channel and basic network packets operations. The functionality of the PHY layer includes packet transmission and reception, clear channel assessment, energy detection and radio state control mechanisms. In addition, it holds and operates with parameters specific for certain radio chips. It includes, for instance, a receiver threshold, carrier sense level and output power range. All parameters related to the PHY-sub-layer are included in the Radio Chip XML specification and can be easily changed in order to model different PHYs.

Access control to the shared communication medium is performed by the MAC sub-layer. This sub-layer acts as an interface between the *Application Layer* and the Radio PHY. MAC is responsible for addressing within the network, data link control, synchronization and quality of service support. Other important function for the MAC protocol is to support and ensuring energy-efficiency and low-level power consumption of the system.

The MAC protocols are implemented either as plug-in libraries or as external Python modules. PASES includes a unified plug-in interface for MAC libraries and provides a template to design and add new MAC models to the framework. The flexibility of such an architecture allows one to design a custom MAC protocol and study its effect on network performance and system power consumption. We implemented a number of the most popular MAC protocols for WSN. Currently these include models of both the beacon-enabled and non-beacon IEEE 802.15.4 MAC [24], S-MAC [25] and TMAC [26] protocols.

Each model of MAC protocol provides a set of common and dedicated services (MAC API). The common API set has to be implemented in each MAC model. It includes basic operations and services on packet transmission and reception, callback functions registration, etc. Dedicated services provide functions specific to each MAC protocols. An example of a dedicated service is a PAN coordinator configuration function in the IEEE 802.15.4 MAC model. Additionally, each MAC includes an external file which contains values for the relevant variables related to the particular MAC model. It may include initial values of the sync periods, beacon intervals, number of enabled retransmissions and others.

The Routing sub-layer is an optional sub-layer logically located on top of the MAC. This layer provides network organization, network adaptation, path discovery and robustness to failure. Currently we support the simple tree-routing algorithm [27] and Ad-hoc On Demand Distance Vector (AODV) routing [28] which is one of the basic routing algorithms in the ZigBee standard. Similarly to the MAC sub-layer models, the routing protocols are implemented as external plug-in modules which can be linked to the simulation infrastructure.

4.7. Network channel model

The **Network Level** (NL) or network channel is an essential part of PASES. This level represents the network communication medium and provides an infrastructure for the network interactions between a number of different nodes. The core functionality of NL is built upon the SystemC Network Simulation Library (SCNSL) [3]. This library provides packet-level network simulation models and independent SystemC interfaces to connect nodes defined at different abstraction levels. Packet forwarding in SCNSL depends on a number of parameters such as relative distance between nodes, transmission power and receiver threshold. This library provides high flexibility, accurate network simulation and high simulation speed.

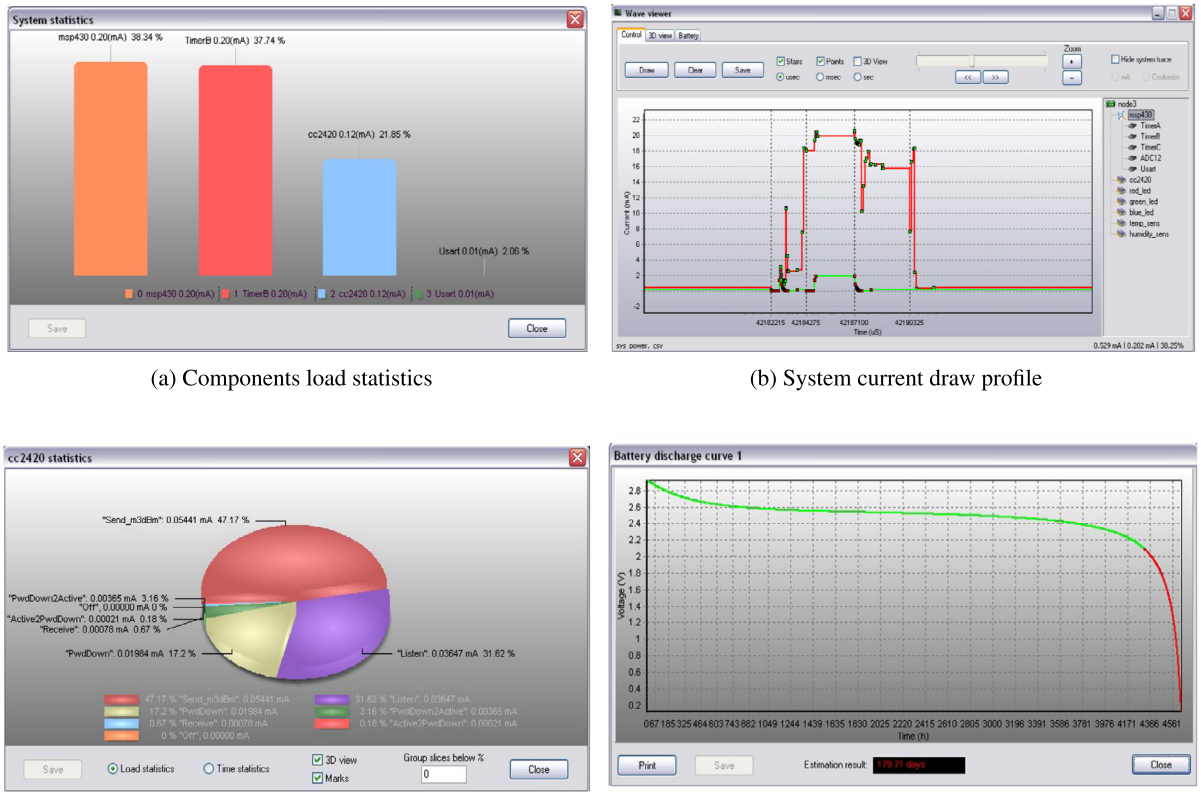
In order to provide flexible and easy-to-use network configuration, PASES utilizes a network XML specification. This file contains information on the node relative locations and network addresses, link packets loss probability, MAC protocol library reference, and many other parameters.

Along with simulation of static networks (nodes with fixed locations), SCNSL provides support for dynamic network topologies. We extended SCNSL with an interface that allows one to assign external mobile scenarios for each simulated node individually. For that purpose, we utilize conventional Python scripts where the user can define node speed, direction and coordinates over simulation time. Python scripts provide a flexible way to specify mobile trajectories at constant or varying speed defined by any equation, look up table or both.

4.8. Power tracer and battery model

Every HW module constituting the node platform model is connected to the system power meter module, which registers the component state transitions at a certain time, calculates the energy consumption of every hardware component at runtime and stores power traces for post processing analysis. The total node power consumption is the sum of the power consumed by each involved component. The framework includes the WaveViewer tool (Fig. 6), used to present graphically the analysis results to the user. System statistics provide information on the relative contribution of each component to the power consumption, as shown in Fig. 6(a). Additionally, the power meter provides precise energy-timing profiling for each HW component individually. Simulation results are stored in csv formatted files individually for each component and can be displayed in the PASES GUI WaveViewer tool. Fig. 6(b) shows one such trace. Likewise, WaveViewer can render the states breakdown for each component involved in the operations of the selected node, as shown in Fig. 6(c).

Another important feature of the WaveViewer is the capability to perform battery lifetime estimation for every node as shown in Fig. 6(d). Lifetime estimation is performed by applying the current draw profile obtained in simulation to the battery model. The runtime battery discharge analysis is considered as a possible task for



(a) Components load statistics

(b) System current draw profile

(c) States breakdown statistics

(d) Battery discharge curve

Fig. 6. PASES post processing GUI.

future work. In PASES we provide two types of different analytical models for post processing lifetime estimation. These include the dynamic State of Charge (SOC) estimation [29] model and a more complex model with rate-capacity and relaxation effects [30,31]. The former model is based on a non-linear equation for State of Charge estimation for the major types of rechargeable battery types, such as Lithium-Ion (Li-Ion), Nickel–Cadmium (NiCd) and Nickel–Metal–Hydride (NiMH) technologies. In spite of certain assumptions taken in this model, such as constant internal resistance, independence from ambient temperature, absence of memory and self-discharge effects, it provides very accurate estimation results.

5. Experimental results

In this chapter we demonstrate the main capabilities of the PASES framework in power-aware design space exploration of WSN applications. Further, we verify its power estimation accuracy by comparing simulation results with reference values obtained on a real test bench. For the sake of presentation, we choose simple scenarios that do not involve a large number of nodes and use a single-hop transmission protocol. However, the framework is capable of supporting a large number of nodes as well as multi-hop networks. We show that, even in the simple case, the framework is able to provide useful trade-off analysis and data for the selection of the best platform and application parameters.

5.1. Application test case

A simple WSN application scenario for ambient temperature monitoring is selected as a reference application. The main focus of this case study is to explore different options in HW/SW and communication implementations, and determine the best combination that meets system requirements at the maximum possible lifetime. The presented scenario involves five wireless nodes organized into a star topology. Four *peripheral* nodes are located within the transmission range of the *central* node (10 m apart). The central node acts both as a network coordinator and as a sink which receives packets from the peripheral nodes. Peripheral nodes, in turn, periodically sample onboard sensors (once in a second), process the obtained values, and send the results to the network sink for data aggregation. Nodes act as Constant Bit Rate (CBR) traffic generators sending packets immediately after retrieving data from the sensors.

In this scenario we evaluate two different HW architectures for the network nodes, i.e., TelosB [18] and a SoC ATmega128RFA1 [20]. The TelosB platform is a popular HW platform upon which we evaluated the PASES modeling accuracy. The second platform is the Atmel SoC which contains an 8-bit ATmega1281 MCU and an AT86RF231 radio transceiver inside a single chip, making this platform very suitable for low-power applications. The energy models for the ATmega128RF components were derived from the device data sheet specification [20]. Table 1 summarizes the current draw and state transition timings of the key operations on both platforms.

By inspecting the table, there is a reasonable expectation that the Atmel platform is more energy efficient that its counterpart, especially with respect to the RF operations which are more energy consuming. However, TelosB is more efficient in terms of CPU power consumption, both in the active and sleep state. It is only by accurately predicting the contributions of each component of the platform that we can determine the best implementation. Even when one platform is consistently superior to another, estimating

Table 1
TelosB and ATmega128RFA1 timing-energy models.

	TelosB	ATmega128RFA1
Operating states:		
Radio Power Down (PwD)	0.01 mA	0.02 mA
Radio RX	19.7 mA	17.6 mA
Radio TX -6 dBm	13 mA	11 mA
CPU Active	2 mA	5 mA
CPU Sleep	0.01 mA	0.02 mA
SPI/IIC	0.1 mA/0.1 mA	0.1 mA
Timer 8 MHz	0.1 mA	0.1 mA
Timer 32 KHz	0.001 mA	0.001 mA
Sensor Active	1 mA	1 mA
State transitions:		
CPU Active to Sleep	0.5 mA/400 us	0.7 ms/200 us
CPU Sleep to Active	0.5 mA/400 us	0.7 ms/200 us
Radio RX to PwD	7 mA/800 us	5 ms/900 us
Radio PwD to RX	6 mA/1920 us	4 ms/1800 us
Radio RX to TX	18 mA/200 us	15 ms/100 us
Radio TX to RX	14 mA/200 us	12 ms/100 us

the quantitative relative difference in terms of node lifetime is difficult to derive directly from the table, and requires a more detailed analysis that can take into account the individual application characteristics. PASES can readily perform this analysis.

Both platforms are very similar in the functionality they offer to the application designers. Thus, it is possible to assemble models of both architectures from the same set of generic HW behavioral components, annotated with different energy-timing values. Technically, the differences in HW are reflected in the dedicated XML platform specifications and corresponding Python annotation files. Hence, both platforms provide the same set of HAL API services for application modeling.

Two SW application models were implemented to be run on the sink and sensor nodes, respectively. The software models involve operations with the MCU, the IIC and the SPI controllers, two Timers, the on board temperature sensor (a model of the SHT11 sensor for both platforms) and the RF transceiver component. All the nodes run on the application duty cycle, periodically switching the radio and the MCU components into lower power states. The sampling rate for sensing operations on the sensor nodes is defined by a peripheral Timer component running at 32.768 Hz. Along with the application, system idle and active periods depend upon the schedule of the chosen MAC protocol and its related parameters.

Along with the two HW architectures, we evaluate two different scheduling-based MAC algorithms. We have chosen the beacon enabled IEEE 802.15.4 MAC [24] and TMAC [32] protocols due to their maturity and popularity for WSN applications. Each model of the MAC is parametrized with a set of specific variables which define scheduling and communication properties. Typically, the available options can be used to customize the MAC synchronization periods, the beacon and frame intervals, the contention periods, the RX/TX buffers lengths, the maximum packet size, and to enable/disable acknowledgments. Thus, the key purpose for this study, along with the HW evaluation, is to find the optimal MAC parameters that provide the best balance between lifetime and other system requirements. In this scenario, we define the packet delivery latency as the most crucial requirement which has to be minimized. Additionally, data delivery reliability is considered as an important objective. Thus, in all cases each successfully delivered packet is confirmed by an acknowledge frame from the receiver. If a packet is not confirmed, the sender resends the data packet again according to the specification of corresponding MAC.

After defining the exploration space in hardware and the application and communication domains, we performed simulations for each HW platform sequentially running each MAC protocol with various parameters. For each simulation case we assume a homogeneous network where all nodes are based on the same HW architecture (either TelosB or ATmega128RF) and run the same MAC protocol algorithm (either IEEE 802.15.4 or TMAC). Additionally, the settings for the radio components for all the nodes are defined as follows: transmission output power for all the nodes was set to -6 dBm, receiver sensitivity threshold to -85 dBm, data rate to 250 Kbps and 16 symbols for Received Signal Strength Indication (RSSI). Further, packet size for all data transmission is fixed to 32 bytes. At the energy source layer, we assume that every node is equipped with two Alkaline AA batteries with 1.5 V nominal voltage and 2500 mAh capacity each. The lifetime prediction is based on the dynamic State of Charge estimation battery model discussed in the previous section.

We present the results in terms of packet latency (throughput is also analyzed, but not presented for brevity). Regarding power consumption, we rather present the amount of current drawn by the different components during the simulation run. We prefer this metric over the more traditional power consumption since current is more easily related to energy, which is the metric that matters in the computation of the system lifetime. Because all simulations are run for 100 s, our metric in this case is also proportional to power. In addition, while the latency is varied, the experiments transfer the same amount of data, making the different solutions comparable. In particular, the current consumption is proportional to energy per data values.

5.1.1. Evaluation of slotted IEEE 802.15.4

In a beacon enabled network, the network (PAN) coordinator periodically transmits beacon packets which other devices use both for synchronization and for determining time periods for transmission and reception of messages. The beacon packets contain information on a *superframe* structure, which defines boundaries for an active and, optionally, for an inactive communication interval. The structure of the superframe is determined by two parameters, the *Beacon Order* (BO) and the *Superframe Order* (SO), which define the length of the active and idle periods, respectively. The setting of BO and SO must satisfy the relationship $0 \leqslant SO \leqslant BO \leqslant 14$. The total length of the superframe (BI) and the length of its active period (SD) are then defined as follows: $BI = aBaseSuperframeDuration * 2^{BO}$, $SD = aBaseSuperframeDuration * 2^{SO}$, where *aBaseSuperframeDuration* is a constant which defines the minimum length of the superframe ($BO = 0$). The standard defines *aBaseSuperframeDuration* as 100 symbols or 16.3 ms at the 250 Kbps data rate. The active interval consists of two periods, the *contention access period* (CAP) and the *optional contention free period* (CFP). The CAP is divided into 16 equally-sized time slots, during which frame transmissions are allowed. To access the network channel during CAP, the nodes utilize a slotted CSMA-CA algorithm. This algorithm relies on two successive CCA operations and backoff periods, where the minimal backoff period is defined by the constant *aUnitBackoffPeriod* (20 symbols). During the inactive period, communication is not allowed and nodes may enter a low-power mode to save battery energy.

We start our exploration from the analysis of the data packets latency at various settings of beacon and superframe intervals. The latency is accounted by the sink SW application as the mean delay of all delivered data packets during a simulation run. For delay analysis, each data frame contains a time stamp included by the sender at the application layer. Fig. 7 presents the obtained data delivery latency for both platforms at various BO and SO values. Here and in the following, BO and SO values are arranged in increasing order of superframe idle communication intervals. As expected, the latency steadily grows along with BO value, which defines the time period between two successive MAC beacons. At small values of beacon interval, more communication periods

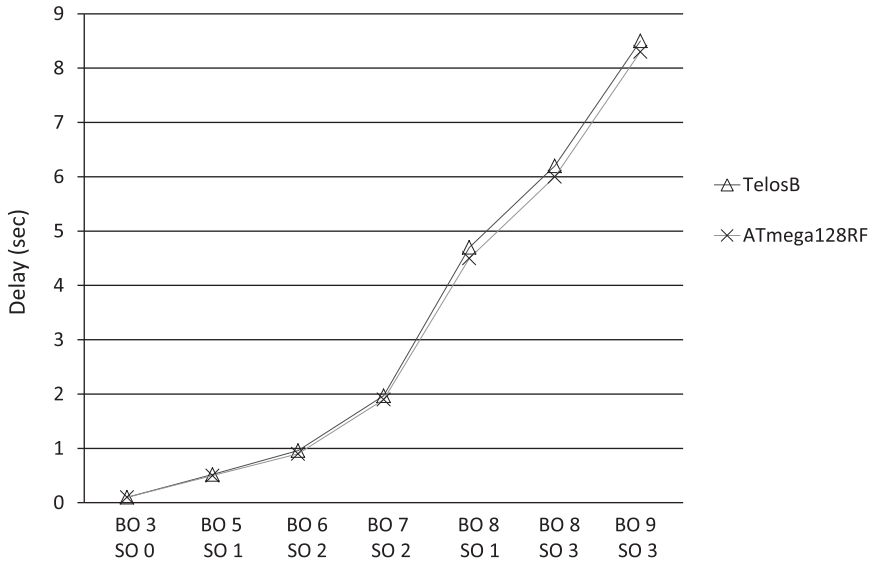


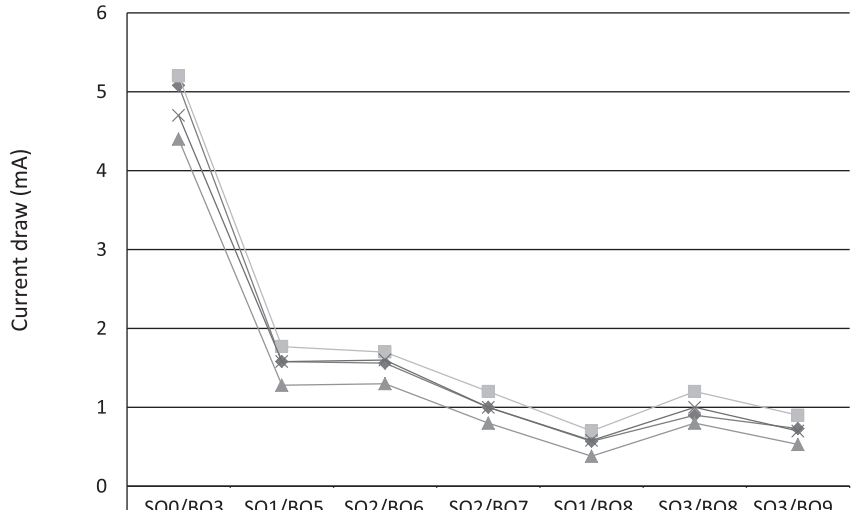
Fig. 7. IEEE 802.15.4 MAC latency for various settings of beacon and superframe intervals.

(CAP) are available in a time unit, thus packets reach their destination with a smaller delay than at long beacon intervals.

Latency is not significantly affected by the choice of platform, since the protocol algorithm and its parameters have a much larger impact. Power consumption instead is different. Fig. 8 shows the obtained current drawn by the sink and the sensor nodes for both investigated HW architectures. In all simulations, the deviation of power consumption between different peripheral nodes in the network is negligible, thus, we assume that all sensor nodes draw an equal amount of current. Further, due to the periodic sensing operations, peripheral nodes consume on average 0.2 mA more than the sink, which does not do any background tasks except for the MAC functionality. Fig. 9 presents the breakdown of the contribution to power consumption for a TelosB peripheral sensor node for the different network parameters. In most cases, the Radio transceiver is the dominant power consumer among other components. However, at BO = 8 and SO = 1, the current draw of the different components is balanced and the contribution of each is around 25%. Finally, Fig. 10 presents the comparison of lifetime estimations for both evaluated HW platforms. Differences are significant, and highlight the impact of the platform on the performance of the application.

5.1.2. Evaluation of TMAC protocol

TMAC (Timeout-MAC) is another popular synchronized MAC protocol specially designed for low power WSN applications [32]. The radio avoids idle listening, which wastes power, by defining an event-driven adaptive schedule with active and idle communication periods. The synchronization of the active/idle intervals between neighboring nodes is accomplished by sending a special packet, called SYNC, which includes the duty cycle timing of the sender, called synchronizer. Neighboring nodes, called followers,



	SO0/BO3	SO1/BO5	SO2/BO6	SO2/BO7	SO1/BO8	SO3/BO8	SO3/BO9
TelosB Sink	5.08	1.58	1.56	1	0.57	0.9	0.73
TelosB Nodes	5.2	1.77	1.7	1.2	0.7	1.2	0.9
Atmega sink	4.4	1.28	1.3	0.8	0.38	0.8	0.53
Atmega nodes	4.7	1.58	1.6	1	0.58	1	0.7

Fig. 8. TelosB and ATmega128RFA1 nodes current draws.

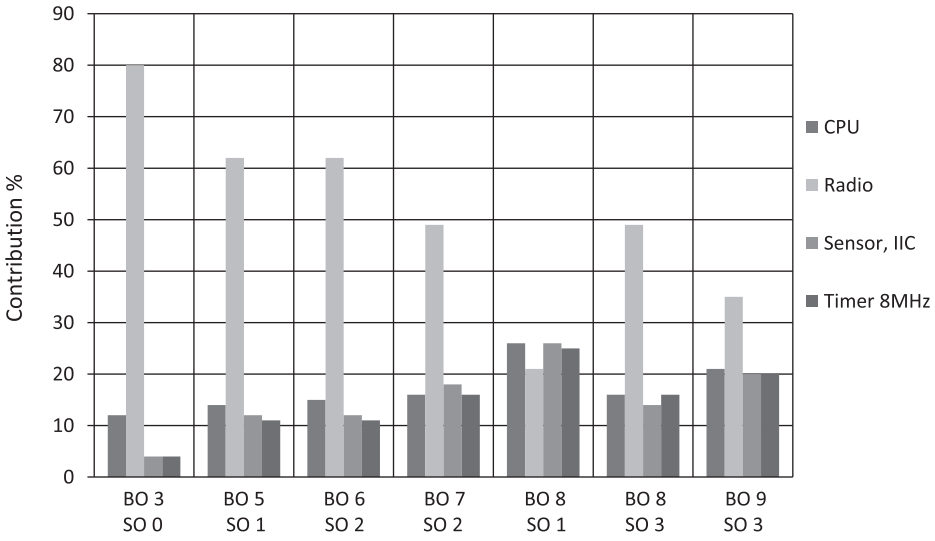


Fig. 9. TelosB node components breakdown.

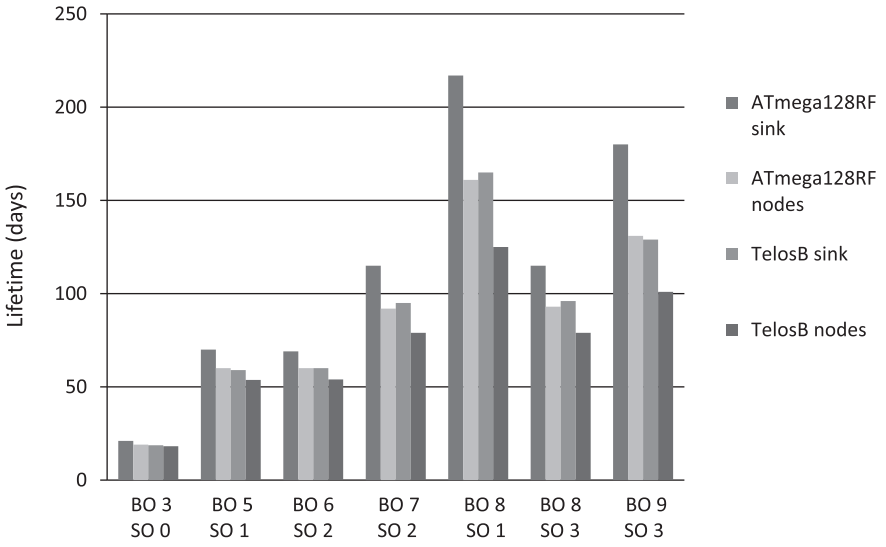


Fig. 10. TelosB and ATmega128RFA1 nodes lifetime.

accept the SYNC and synchronize with the obtained schedule, in order to wake up and communicate at the same time. Further, each node may accept and follow a number of different schedules, acting as a connector between network clusters with different synchronizers. Unicast message passing in TMAC relies on the RTS (Request To Send) and CTS (Clear To Send) handshaking process [32] implemented in CSMA-CA. Additionally, successfully received data frames are optionally confirmed by the receiver with acknowledgment packets. Thus, a typical unicast data message consists of an RTS/CTS/DATA/ACK packet exchange sequence.

The evaluation of the TMAC protocol focuses on the same objectives as above for the IEEE 802.15.4. We aim to discover the optimal protocol settings including scheduler intervals (frame intervals FI), activation timeout (TA) and set of activation events in order to achieve the minimal level of data delivery at the maximum possible network lifetime. We performed a series of simulations for both HW platforms with various FI and TA at a fixed set of activation events which included all events mentioned above. Fig. 11 presents the obtained packet latency at different intervals of FI and TA for both architectures. The delivery delay linearly depends on the length of the frame interval, and is essentially identical for both architectures. The obtained current draw for both the TelosB and Atmel platforms is shown on Fig. 12. We observe that the TMAC protocol does not provide any considerable energy saving at extended frame intervals, due to the traffic adaptive length of the active interval. All the nodes remain active until they completely outflow all awaiting data frames, that leads to the long active intervals where most of the energy is spent.

We can use PASES to explore alternatives and optimization strategies. For example, in order to reduce the amount of network transmissions and consequently scale back the length of the active intervals, frame aware duty cycling can be applied to the SW application of the peripheral nodes. For instance, instead of creating a single packet for every sensing sample and buffer them locally, data can be aggregated into a single frame during the idle interval and sent once when the active period begins. Another optimization approach is to reduce the sensing rate to conform with the MAC frame interval period. Fig. 13 shows the results in terms of current draws on TelosB nodes only, when the sensing rate is varied according to the MAC FI.

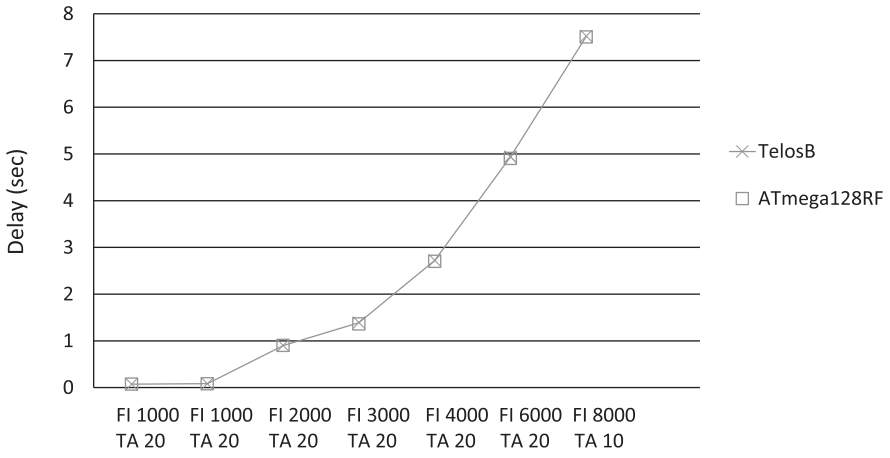
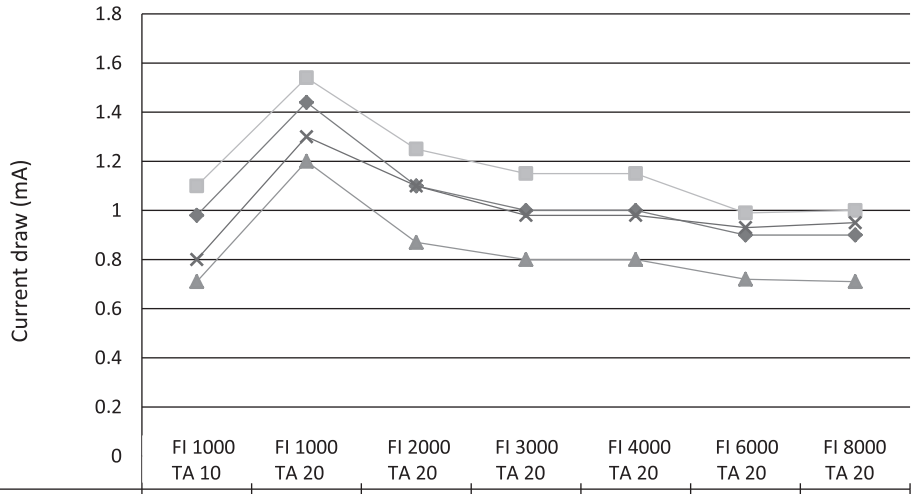


Fig. 11. TMAC packet delivery latency.



	Fl 1000 TA 10	Fl 1000 TA 20	Fl 2000 TA 20	Fl 3000 TA 20	Fl 4000 TA 20	Fl 6000 TA 20	Fl 8000 TA 20
TelosB sink	0.98	1.44	1.1	1	1	0.9	0.9
TelosB nodes	1.1	1.54	1.25	1.15	1.15	0.99	1
Atmega sink	0.71	1.2	0.87	0.8	0.8	0.72	0.71
Atmega nodes	0.8	1.3	1.1	0.98	0.98	0.93	0.95

Fig. 12. Current draw.

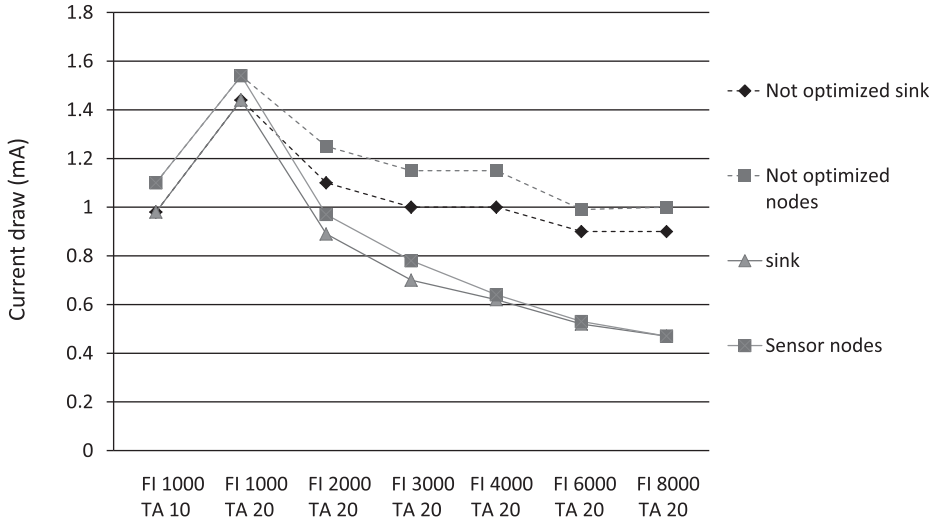


Fig. 13. TelosB current draw with optimized rate.

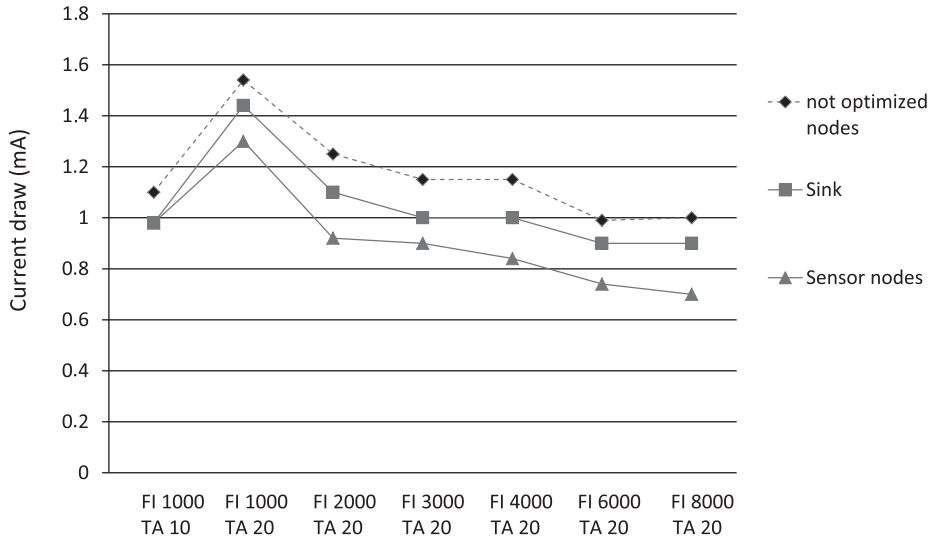
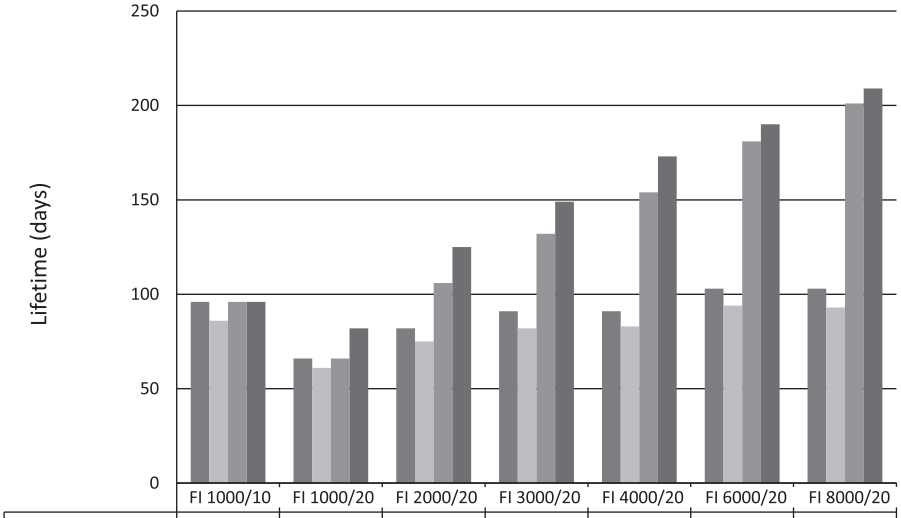


Fig. 14. TelosB current draw with overhearing avoidance optimization.



	FI 1000/10	FI 1000/20	FI 2000/20	FI 3000/20	FI 4000/20	FI 6000/20	FI 8000/20
initial sink	96	66	82	91	91	103	103
initial sensor nodes	86	61	75	82	83	94	93
Sink	96	66	106	132	154	181	201
Sensor nodes	96	82	125	149	173	190	209

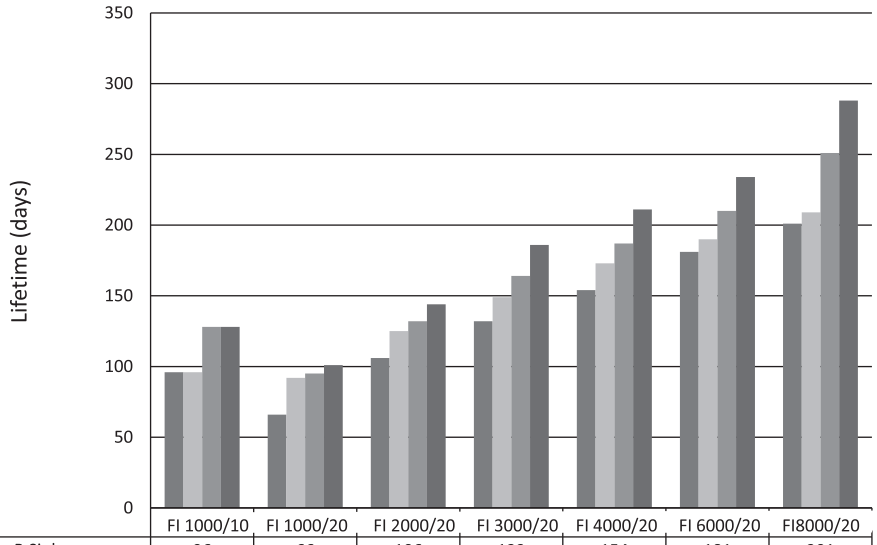
Fig. 15. Estimated lifetime for TelosB nodes.

Another optimization can be made on the MAC algorithm itself, by avoiding certain activation events upon which nodes prolong their active communication intervals. Since in the presented scenario only unidirectional data traffic is considered, overhearing avoidance can be applied for the sensor nodes. This allows a node to reduce energy waste during active periods, when the outgoing buffer is empty while the node still overhears communication activity of its neighbors, forcing it to keep the radio on. By applying overhearing avoidance, each sensor node can turn off its radio and go to the idle mode immediately after it completes transmission of the last packet awaiting in the queue. Fig. 14 presents the current draws of TelosB nodes at a constant sensing rate (1 Hz) using the overhearing avoidance optimization for the peripheral nodes. In this case sensor nodes consumes different amount of energy at each active transmission period due to varied length of communication periods. For this reason, Fig. 14 presents mean values of drawn current among all the sensor nodes.

Combining both optimization approaches (varied sampling rate and overhearing avoidance) together, the network lifetime can be increased by up to 50% (at FI equal to 8000 ms) with respect to the initial settings, as shown on Fig. 15. It is important to note that the applied optimization provide considerable energy saving while not affecting packet latency delay. Finally, Fig. 16 presents a comparison of the estimated lifetime for TelosB and Atmel SoC platforms with both optimization techniques applied.

5.1.3. Discussion

In this study we demonstrated that even a straightforward application may have a rather wide design space and various configuration options which directly affect the network performance



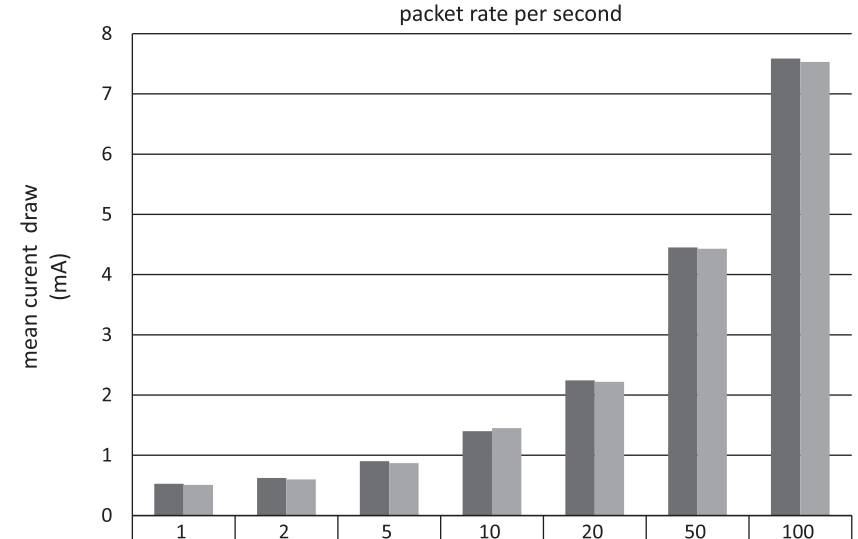
	FI 1000/10	FI 1000/20	FI 2000/20	FI 3000/20	FI 4000/20	FI 6000/20	FI8000/20
TelosB Sink	96	66	106	132	154	181	201
TelosB Sensors	96	92	125	149	173	190	209
ATmega128RF Sink	128	95	132	164	187	210	251
ATmega128RF Sensors	128	101	144	186	211	234	288

Fig. 16. Lifetime comparison of TelosB and Atmel SoC platforms.

and lifetime. For both estimated MAC protocols, the mean current draw of the Atmel platform is on average 18% less than the TelosB under the same conditions. As a consequence, the Atmel SoC offers a more efficient HW architecture for ultra low power applications. The evaluation of the two different MAC models aimed to select the optimal configuration options for each, which provide the maximum possible lifetime at minimal level of data delivery latency. We have shown that both evaluated communication protocols offer similar functionality and both rely on a centralized synchronization mechanism. Both protocols provide very similar energy efficiency at latency intervals below 5 s. However, at longer periods, the TMAC protocol using the proposed optimizations outperforms the IEEE802.15.4 standard in terms of energy efficiency. Thus, our design space exploration shows that the TMAC algorithm, in conjunction with an ATmega128RFA1-based platform, provides a more efficient solution for WSN applications which can tolerate high latency.

5.2. Energy model verification

In order to obtain the power consumption reference values with which simulation results can be compared to, we have set up a test bench consisting of three TelosB nodes. Two of those act as CBR traffic generators and one as a sink connected to a PC for data



	1	2	5	10	20	50	100
PASES	0.527	0.622	0.902	1.4	2.243	4.451	7.588
Measurements	0.507	0.6	0.87	1.45	2.22	4.43	7.53

Fig. 17. Current draw estimation vs measurement.

gathering. The sender nodes utilize a simple power saving technique that controls the operating states of the MCU and RF subsystems. Whenever the application has a packet to send, the RF component is switched from the idle to the active state. Upon completion of the transmission, the radio is turned back to the idle mode. Hence, the power consumption of the nodes is essentially determined by the transmission rate of the application that defines how often the radio is required to be in the active state. All communications in our experiments and simulations are based on a unslotted IEEE802.15.4 CSMA/CA algorithm. The software applications and the MAC algorithm for both the sink and the sender nodes were implemented in TinyOS 2.1.1. The current load profile sensor nodes were obtained by logging the voltage drop across resistor connected between the board and power source. The collected voltage profile was subsequently converted to the current values. We carried out simulations and experiments with different transmission rates ranging from one to one hundred packets per second. The obtained simulation results and measured current draw are shown on Fig. 17.

The comparison shows that PASES provides a high level of accuracy in current draw estimation with an average error equal to 3%. This accuracy was achieved by using a fine grained energy model for the peripheral components, which have been characterized through a set of micro-benchmark experiments that isolated current consumption and state transitions of each device on the target board.

6. Conclusion

In this paper we presented PASES, a SystemC-based framework for design space exploration of networked embedded systems and WSNs. PASES provides detailed modeling and precise power consumption evaluation and lifetime estimation of target applications allowing the designer to explore various options and details in SW/HW and communication domains. PASES is extensible and customizable, and allows designers to easily trade off accuracy versus performance, with deployments that can include tens to hundreds of nodes, communicating over multi-hop networks. The framework, together with a comprehensive library of characterized components and communication protocols, and a number of example applications (including those presented in this paper), is available for download at http://eecs.disi.unitn.it/pases.

References

[1] Network simulator 2, Accessed December 2012. <http://nsnam.isi.edu/nsnam/index.php/MainPage>.

[2] Omnet++, Accessed December 2012. <http://www.omnetpp.org/>

[3] F. Fummi, D. Quaglia, and F. Stefanni, A SystemC-based framework for modeling and simulation of networked embedded systems, in: Proceedings of the Forum on Specification, Verification and Design Languages, FDL 2008, Stuttgart, Germany, September 23–25, 2008, pp. 49–54.

[4] Thomas R. Henderson, Mathieu Lacage, George F. Riley, Network simulations with the ns-3 simulator, in: Proceedings of ACM SIGCOMM08, Seattle, WA, USA, August 17–22, 2008.

[5] OPNET network simulator, Accessed December 2012.<http://www.opnet.com/>

[6] Qualnet, Accessed December 2012. <http://www.scalable-networks.com/content/products/qualnet>.

[7] Athanassios Boulis, Castalia: revealing pitfalls in designing distributed algorithms in wsn, in: Proceedings of the 5th International Conference on Embedded Networked Sensor Systems, SenSys'07, Sydney, Australia, November 6–9, 2007, pp. 407–408.

[8] A. Köpke, M. Swigulski, K. Wessel, D. Willkomm, P.T. Klein Haneveld, T.E.V. Parker, O.W. Visser, H.S. Lichte, S. Valentin, Simulating wireless and mobile networks in OMNeT++ the MiXiM vision, in: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops, Simutools '08, Marseille, France, March 3–7, 2008, pp. 71:1–71:8

[9] D. Weber, J. Glaser, S. Mahlknecht, Discrete event simulation framework for power aware wireless sensor networks, in: Proceedings of the 5th IEEE International Conference on Industrial Informatics, Vienna, Austria, July 23–27, 2007, pp. 335–340.

[10] A. Fraboulet, G. Chelius, E. Fleury, Worldsens: development and prototyping tools for application specific wireless sensors networks, in: Proceedings of the 6th International Symposium on Information Processing in Sensor Networks, IPSN '07, Cambridge, MA, USA, April 25–27, 2007, pp. 176–185.

[11] Mohammed Baz and David A.J. Pearce, An introduction to DANSE, in: Proceedings of the 12th Annual PostGraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting, PGNet '11, Liverpook, UK, June 27–28, 2011.

[12] Enrico Perla, Art Ó Catháin, Ricardo Simon Carbajo, Meriel Huggard, Ciarán Mc Goldrick, PowerTOSSIM z: realistic energy modeling for wireless sensor network environments, in: Proceedings of the 3rd ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks, PM2HW2N'08, Vancouver, BC, Canada, October 31, 2008, pp. 35–42.

[13] B.L. Titzer, D.K. Lee, J. Palsberg, Avrora: scalable sensor network simulation with precise timing, in: Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks, IPSN'05, Los Angeles, CA, USA, April 25–27, 2005, pp. 477–482.

[14] J. Polley, D. Blazakis, J. McGee, D. Rusk, J.S. Baras, ATEMU: a fine-grained sensor network simulator, in: Proceedings of the First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, SECON '04, Santa Clara, CA, USA, October 4–7, 2004, pp. 145–152.

[15] Douglas Densmore, Roberto Passerone, Alberto L. Sangiovanni-Vincentelli, A platform-based taxonomy for ESL design, IEEE Design and Test of Computers 23 (5) (2006) 359–374.

[16] Alessandro Pinto, Alvise Bonivento, Alberto L. Sangiovanni-Vincentelli, Roberto Passerone, Marco Sgroi, System level design paradigms: Platform-based design and communication synthesis, ACM Transactions on Design Automation of Electronic Systems 11 (3) (2006) 537–563.

[17] Philip Levis, Nelson Lee, Matt Welsh, David Culler, TOSSIM: accurate and scalable simulation of entire TinyOS applications, in: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, SenSys '03, Los Angeles, CA, USA, November 5–7, 2003, pp. 126–137.

[18] Memsic Inc., MICAz, MICA2 - wireless measurement system, telosb mote platform, Accessed December 2012. <http://www.memsic.com>.

[19] O. Landsiedel, K. Wehrle, S. Gotz, Accurate prediction of power consumption in sensor networks, in: Proceedings of the Second IEEE Workshop on Embedded Networked Sensors, EmNetS '05, Sydney, Australia, May 30–31, 2005, pp. 37–44.

[20] Atmel Corporation, San Jose, CA, USA, 8-bit AVR Microcontroller with Low Power 2.4 GHz Transceiver for ZigBee and IEEE 802.15.4, 2011.

[21] Luca P. Carloni, Fernando De Bernardinis, Alberto L. Sangiovanni-Vincentelli, Marco Sgroi. The art and science of integrated systems design, in: Proceedings of the 28th European Solid-State Circuits Conference, ESSCIRC 2002, Firenze, Italy, September 2002, pp. 19–30.

[22] Frank Ghenassia, Transaction Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems, Springer, 2005.

[23] Texas Instruments, MSP430x1xx Family, Users Guide, SLAU094F, 2011.

[24] Jose A. Gutierrez, Edgar H. Callaway, Raymond L. Barrett, Low-rate wireless personal area networks: enabling wireless sensors with IEEE 802.15.4, Technical Report, Institute of Electrical & Electronics Engineers (IEEE), 2004.

[25] Wei Ye, J. Heidemann, D. Estrin, An energy-efficient MAC protocol for wireless sensor networks, in: Proceedings of the Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies, volume 3 of INFOCOM '02, New York, NY, USA, June 25–27, 2002, pp. 1567–1576.

[26] Joseph Polastre, Jason Hill, David Culler, Versatile low power media access for wireless sensor networks, in: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, SenSys '04, Baltimore, MD, USA, November 3–5, 2004, pp. 95–107.

[27] Mohamed F. Younis, Moustaf Youssef, Kaled A. Arisha, Energy-aware routing in cluster-based sensor networks, in: Proceedings of the 10th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOT'02, Fort Worth, TX, USA, October 11–16, 2002, pp. 129–136.

[28] C.E. Perkins, E.M. Royer, Ad-hoc on-demand distance vector routing, in: Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications, WMCSA '99, New Orleans, LA, USA, February 25–26, 1999, pp. 90–100.

[29] O. Tremblay, L.-A. Dessaint, A.-I. Dekkiche, A generic battery model for the dynamic simulation of hybrid electric vehicles, in: Proceedings of the IEEE Vehicle Power and Propulsion Conference, VPPC'07, Arlington, TX, USA, September 9–12, 2007, pp. 284–289.

[30] Marijn R. Jongerden, Boudewijn R. Haverkort, Which battery model to use?, IET Software 3 (6) (2009) 445–457

[31] Debashis Panigrahi, Sujit Dey, Ramesh Rao, Kanishka Lahiri, Carla Chiasserini, Anand Raghunathan, Battery life estimation of mobile embedded systems, in: Proceedings of the The 14th International Conference on VLSI Design, VLSID '01, Bangalore, India, January 3–7, 2001.

[32] Tijs van Dam and Koen Langendoen, An adaptive energy-efficient MAC protocol for wireless sensor networks, in: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, SenSys'03, Los Angeles, CA, USA, November 5–7, 2003, pp. 171–180.

Ivan Minakov is a research scientist in the Department of Information Engineering and Computer Science at the University of Trento, Italy. His main research interests focus on design methodologies, modeling and power analysis of wireless sensor networks and embedded systems. Minakov received a PhD in electrical engineering from the University of Trento in 2012.

Roberto Passerone is Assistant Professor at the Department of Information Engineering and Computer Science at the University of Trento, Italy. He received his MS and PhD degrees in Electrical Engineering and Computer Sciences from the University of California, Berkeley, in 1997 and 2004, respectively. Before joining the University of Trento, he was Research Scientist at Cadence Design Systems. Prof. Passerone has published numerous research papers on international conferences and journals in the area of design methods for systems and integrated circuits, formal models and design methodologies for embedded systems, with particular attention to image processing and wireless sensor networks. He was track chair for the Real-Time and Networked Embedded Systems at ETFA from 2008 to 2010, and general and program chair for SIES 2010 and 2011. He was guest editor of the Transactions on Industrial Informatics for numerous special issues on embedded systems.