# Virtual Platform-Based Design Space Exploration of Power-Efficient Distributed Embedded Applications

PARINAZ SAYYAH, Harman Becker Automotive Systems, Italy
MIHAI T. LAZARESCU, Politecnico di Torino, Italy
SARA BOCCHIO, STMicroelectronics, Italy
EMAD EBEID, Aarhus University, Denmark
GIANLUCA PALERMO, Politecnico di Milano, Italy
DAVIDE QUAGLIA, University of Verona and EDALab s.r.l., Italy
ALBERTO ROSTI, STMicroelectronics, Italy
LUCIANO LAVAGNO, Politecnico di Torino, Italy

Received March 2014; revised October 2014; accepted January 2015

This work was supported by the EU FP7 COMPLEX project.
Authors' addresses: P. Sayyah, Harman Becker Automotive Systems s.r.l., Corso Unione Sovietica 612/15/A, I-10135 Torino (TO), Italy; email: parinaz.sayyah@harman.com; M. T. Lazarescu and L. Lavagno, Politecnico di Torino, Dipartimento di Elettronica e Telecomunicazioni, Corso Duca degli Abruzzi 24, I-10129 Torino (TO), Italy; emails: {mihai.lazarescu, luciano.lavagno}@polito.it; S. Bocchio and A. Rosti, STMicroelectronics, Via C. Olivetti 2, I-20864 Agrate Brianza (MI), Italy; emails: {sara.bocchio, alberto.rosti}@st.com; E. S. M. Ebeid, Department of Engineering, Finlandsgade 22, 8200 Aarhus N, Denmark; email: esme@eng.au.dk; G. Palermo, Politecnico di Milano, Dipartimento di Elettronica, Informazione e Bioingegneria, Via Ponzio 34/5, I-20133 Milano (MI), Italy; email: gianluca.palermo@polimi.it; D. Quaglia, Università di Verona, Dipartimento di Informatica, Strada le Grazie 15, I-37134 Verona (VR), Italy; email: davide.quaglia@univr.it.

## 1. INTRODUCTION

According to several reports, market opportunities offered by the Internet of Things and Machine-to-Machine Communication will dramatically grow in the following years, covering applications in various domains, such as healthcare, smart energy and water management, building automation, and more [Mattern and Floerkemeier 2010]. Embedded systems will be fundamental building blocks of these applications, working in a fully distributed environment where communications play a fundamental role.

Embedded systems are becoming ever more complex and provide an increasing number of features that can be exploited by software components to fulfill the application requirements. In particular, power management is crucial to increase the battery lifetime or to efficiently exploit energy harvesting from the environment. Power-saving strategies of course must take into account the real-time constraints of the application, such as the processing delay of asynchronous events or the sampling rate of analog inputs.

Network conditions may also weigh in for communication-intensive applications. For example, the assessment of the congestion of a shared networking channel can be used to dynamically adapt the behavior of the wireless sensor nodes of the network to optimize their power consumption. Basically, there is no point in collecting and transmitting data when the network cannot deliver them. A functional and performance model of the network communications allows fast investigation of these effects during application development. Thus, the scalability of the network can be assessed at design time, avoiding expensive deployments and in-field measurements and testing.

For these reasons, a virtual platform is needed for the design space exploration (DSE) of networked embedded systems. We believe, for reasons that will be explained more in detail in the rest of the article, that it should contain the following key elements:

—Behavioral and power models of the hardware components, such as the CPU, the memory, and the peripherals. The behavioral level provides a good level of simulation efficiency and intellectual property protection while accurately modeling the power effects at runtime.
—A cycle-accurate instruction set simulator (ISS) that can execute the unmodified application binary in close interaction with the hardware component models
—Primitives to model and simulate the network in terms of packet transmission and reception, channel effects, stochastic models of concurrent traffic sources, packet collisions, and protocol simulation
—Model-driven design for application specification and implementation
—Automated generation of the application source and binary code to allow DSE automation

The article describes a development flow that uses a SystemC-based virtual platform supporting the design of energy-efficient distributed embedded applications. The approach provides the following key features:

—Model-driven design of the application, including the automated generation of the software components
—Use of a power-aware SystemC model of a system-on-chip (from STMicroelectronics) that accurately reproduces the energy management strategy effects during the DSE
—Modeling both realistic network scenarios and network nodes directly in SystemC, without the need for cosimulation with external tools

These features are demonstrated in the context of design space exploration for energy optimization of a wireless body area network. Our experiments take into consideration both the network conditions and the application requirements.

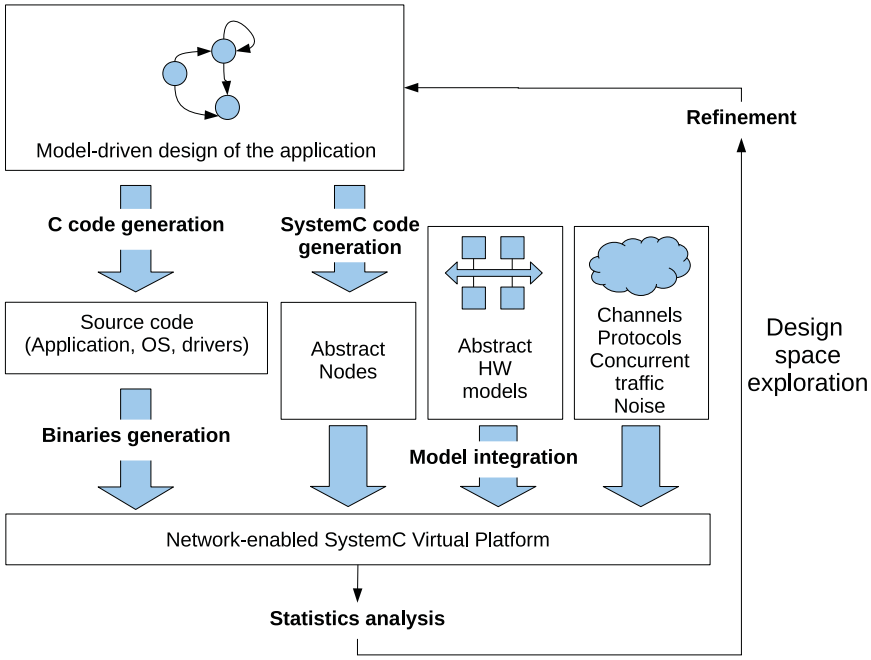



Fig. 1. Proposed design space exploration flow for distributed embedded applications.

Figure 1 shows the proposed design space exploration flow for distributed embedded application design that is built around the network-enabled SystemC virtual platform. The platform provides a realistic description of the network environment that is used by its component nodes. Effects such as packet collisions on shared channels, packet loss due to queue overflows within nodes, transmission delays, bit corruptions, and signal energy loss are also modeled.

The behavior of the network nodes is described using platform-independent State-Charts (using the StateFlow tool). They are used to generate both the application C-language source code (using the code generation tools provided by The Mathworks) and the network node SystemC behavioral model (using the HIFSuite toolset [Lazarescu et al. 2012]).

The source StateFlow model is thus translated:

—for a few selected nodes, into C and then binary code to be executed on the ISS of the virtual platform, together with the operating system and the drivers. The software components interact at runtime with the hardware components that are modeled at the SystemC Transaction Level (TLM) to reduce the simulation time. The hardware component models include the power effects of the software-defined energy-saving strategies to allow an accurate evaluation of the power consumption during the DSE.
—for the rest of the nodes, directly in SystemC, in order to ensure fast simulation even with very large networks.

The ISS can profile the execution and the power consumption of any software on the node platform, even if some source code (e.g., the RTOS) is not available. The binary code does not need to be changed or annotated. Furthermore, the application does not need to be manually rewritten for the target platform, because this is handled by the StateFlow code generation tools.

The configuration parameters of the application can be tuned by performing comparative performance analysis of simulation runs with different environmental conditions (an example is shown in Section 7.3). The virtual platform allows one to obtain statistics on both functional and nonfunctional properties of the distributed system (e.g., responsiveness and energy consumption). If the application requirements are not met, the parameters can be refined in a new exploration cycle.

Although modern ISSs are fast, their use does not scale well for simulations with a large number of nodes. For this reason, the proposed approach allows one to model the nodes at different levels of detail. For instance, the behavior of the application running on a node can be reproduced with instruction-level accuracy on the ISS to study its power-saving features, while other network nodes can be modeled at a pure behavioral level using the code generated for SystemC. Also, the proposed SystemC virtual platform can be interfaced with different ISSs to support different hardware platforms.

Hardware-in-the-loop simulations can also capture architectural and power runtime effects with a limited simulation slowdown [Mozumdar et al. 2010]. However, they require the instrumentation of the code running on the nodes, and the architectural and power modeling is less accurate than what is provided by an ISS.

This development flow has been used to develop a wireless body area network application based on the SPINE framework [Gravina et al. 2008] and on the ReISC SoC by STMicroelectronics, in the context of the European project COMPLEX[1] [Grüttner et al. 2013]. The ReISC SoC provides complex and effective power management strategies, but they are difficult to exploit by the developers without a network-enabled simulation platform. Therefore, our proposed approach can have a significant impact on the adoption of the SoC and on the quality of the applications developed.

The rest of the article is organized as follows. Section 2 reviews the published papers addressing related topics. Section 3 describes the reference design scenario. Sections 4 and 5 describe the modeling of the software and hardware components, respectively. Section 6 describes the modeling of the network aspects. Section 7 presents different testing scenarios. Section 8 shows how the design space exploration can be performed using the virtual platform. Section 9 concludes the article.

## 2. RELATED WORK

The design of distributed embedded applications may often require simulation at both the node and the network levels early in the design phase.

Many simulators exists for this purpose, such as NS-2/3 [McCanne et al. 1989], TOSSIM [Levis et al. 2003], EmStar [Girod et al. 2004], OPNET [Chang 1999], OMNeT++ [Varga and Hornig 2008], J-Sim [Miller et al. 1997], Atemu [Polley et al. 2004], Avrora [Titzer et al. 2005], Cooja [Osterlind et al. 2006], Viptos [Cheong et al. 2005], and Pawler [Simon et al. 2003]. Du et al. [2010] provide a detailed analysis of the field. However, most simulators are not well suited for both node and network simulation, since they usually model well either the communication channel or the node hardware.

The communication channel can be modeled in C, C++, or Java using either discrete event-based network simulators, such as NS-2/3, OPNET, OMNeT++, and TOSSIM, or generic ones, such as Matlab [The Mathworks 1998].

The node can be simulated with high-level modeling tools (e.g., Stateflow from The Mathworks, which models behaviors using StateCharts) or with sensor network-specific simulators, such as Avrora, Cooja, Atemu, Viptos, and Pawler. The latter usually provide a very abstract hardware model that may fail to capture significant details with respect to an HDL simulation. Moreover, some of these tools are limited to specific HW/SW platforms (e.g., AVR based) or to specific programming languages (e.g., nesC).

[1]FP7 IP 247999 COMPLEX: https://complex.offis.de/.

Cosimulation attempts to overcome these limitations by using different tools to simulate the different domains (e.g., digital hardware and network). They run independently and synchronize through some form of interprocess communication. Mulas et al. [2010] demonstrate how the node energy consumption can be reduced by scaling down the CPU clock frequency during periods of network congestion that are detected based on the length of the transmission queue. However, process synchronization slows down the cosimulation [Lora et al. 2014], making it less suitable for efficient DSE, which requires the simulation of a very large number of configurations.

For an efficient optimization of both node-level and network-level features, we need to simulate all domains using a single tool and description language. In this way, we avoid the synchronization overhead by using a single executable model for the DSE [Palermo et al. 2009]. For this purpose, system modeling languages such as SystemC [Liao et al. 2002] are increasingly used for SoC design, since they allow both large-scale and small-scale performance simulation and optimization. SystemC can be used to flexibly model the hardware, the application software, and the network components at different levels of abstraction. Also, it can be used to create and efficiently use library components for Transaction Level Modeling (TLM), Simulation, and Verification.

The method presented by Streubühr et al. [2011] uses SystemC to estimate the power consumption of heterogeneous multiprocessor SoCs at the Electronic System Level. The simulation models can be extended to account for the power states that are used for the dynamic evaluation of the energy consumption of the system.

In this work, we use SystemC to (1) encapsulate the ISS for the software, (2) model the hardware, and (3) describe the network using the SystemC Network Simulation Library (SCNSL) [Quaglia and Stefanni 2013]. This allows us to provide both node-level and network-wide performance analysis in a single simulation environment, without requiring any other tool for system and network modeling or cosimulation.

Our simulation flow uses a model-based approach to achieve the best of both domains in terms of speed and accuracy. We use a *single StateFlow application model* to drive:

(1) executable code generation for a fast yet accurate SystemC hardware model of *one node* (or a few nodes), which includes an ISS and a power model of the hardware node platform. We consider that this is the highest abstraction level that allows us to obtain sufficiently accurate power simulation results for the hardware blocks of the node platform when operated under an aggressive power management strategy. Our case study involved only one node modeled at this level, but more nodes can be simulated in this way if needed for other applications.
(2) an abstract SCNSL network model that is accurate enough to capture network access issues and transmission errors.

It is worth noting that we are using *the same Stateflow model* of the WSN application to generate both the code running on the ISS-based node model and the pure behavioral model of each node instance running in the SCNSL framework, while preexisting methods require one to *develop and maintain two independent source code projects* (one for the network simulation and one for the ISS), both representing the same functionality.

Our approach is similar to IDEA1 [Du et al. 2011], but we improve *at the same time* both the *scalability* and the *accuracy* of the analysis through a low-level node model (based on an industrial Virtual Platform SoC model) included in a network of SystemC-modeled nodes at the application and protocol stack levels.

To the best of our knowledge, this model-driven application design and holistic simulation approach is the first to *generate a complete system model* that can be simulated, optimized, and verified at both the network and the node levels. Moreover, the methodology can be extended to include any high-level modeling flow that can generate a suitable SystemC model.

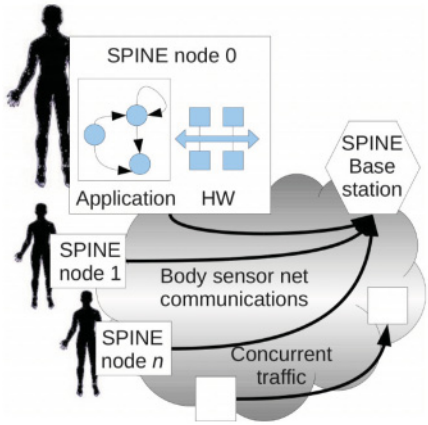



Fig. 2. Wireless body sensor network application.

The effectiveness of the proposed solution is shown through the design of energy-saving strategies for wireless sensor networks. Most of the existing approaches focus on communication energy, since this ranks among the highest sources of energy consumption at the node level [Croce et al. 2008]. These techniques can be used in the proposed virtual platform, and we explore novel ways to reduce the computation energy consumption by using knowledge about the communication status.

## 3. REFERENCE DESIGN SCENARIO

We use a realistic distributed embedded application, from the health monitoring domain, to illustrate our proposed design flow. The application includes real-time data acquisition (from sensors attached to the human body) and preprocessing on the node itself, and communication over a shared channel to a remote server.

Figure 2 shows the application architecture. Wearable nodes with suitable sensors are attached to specific body locations. Data, either raw or preprocessed on the nodes, is sent to the Base Station (BS) for further processing, recording, displaying, and alerting on specific patterns and events (e.g., motion, immobility, fall).

The BS can receive concurrent communications over a shared radio channel from several sensor nodes, which can lead to channel congestion if left unmanaged. Also, adapting the node radio traffic to channel status can save energy on the node and increase its service time, as will be discussed in detail later.

The wireless nodes follow the open-source Signal Processing in Node Environment (SPINE) specification [Gravina et al. 2008]. It supports flexible and distributed signal processing for wireless body sensor network applications through a set of customizable functions for data acquisition, processing, and communication.

In the following, we will focus on the analysis and optimization of performance and energy consumption of the SPINE node, since we assume that the BS is much less resource constrained. We will also show how the platform-dependent simulation of the entire application increases the accuracy of the power analysis.

### 3.1. Base Station

The BS runs a fall detection algorithm that processes the data sent by all wearable SPINE nodes. The detection algorithm can be in one of the following states:

—*Person standing*, no alarm
—*Possible person fall*, set prealarm

—*Possible person impact*, no alarm
—*Wait for reading stabilization*, no alarm
—*Person not moving*, send alarm

Falls are signaled when the acceleration rises above a given threshold.

The BS algorithm was tested using emulated sensor node samples. They were obtained from a dataset provided by the German Aerospace Center, which includes patterns for different human physical activities, including falls.

### 3.2. Wearable SPINE Nodes

The basic operation of a wearable SPINE node consists of:

—processing the configuration packets received from the BS;
—performing the configured computations;
—transmitting the results to the BS.

The SPINE specification provides the application developer with several functions to build the monitoring application, such as:

—reading and local buffering of data from various sensors (e.g., temperature, acceleration, blood oxygenation level, hearth pulse rate);
—processing the buffered data through filters, threshold detection, and other mathematical functions;
—sending data packets to the BS either periodically or upon event detection.

The BS can configure at runtime all SPINE functions through specific packets. This includes the selection of parameters such as the sampling period of each sensor and the buffer sizes, as well as the list of features to extract (e.g., max, min, median). The last packet in the configuration sequence provides the list of tasks to activate (e.g., sampling, feature computation, radio packet dispatch).

In our case study, raw acceleration data is read on three axes and filtered using a Butterworth filter to reduce the noise. The acceleration magnitude is calculated from the three components and is sent to the BS in a data packet over a shared wireless channel.

## 4. SOFTWARE MODELING

Platform-independent model-driven design and automated code generation can address the complexity of the development of distributed embedded applications. They simplify the process of debugging and refining the software under development, enabling faster design cycles and higher degrees of flexibility and reusability.

### 4.1. Model-Driven Design

High-level modeling simplifies software development by replacing coding-debugging iterations with modeling and simulation activities in a graphical environment.

Our design flow starts by defining platform-independent Stateflow models that can be converted, by means of target-specific code generators, into:

—SystemC behavioral models for platform-independent simulation and functional verification of the whole distributed embedded application in the SCNSL framework;
—platform-dependent application C code to be compiled and executed by the ISS of the ReISC SoC virtual platform.

A simulation model of the SPINE virtual machine was implemented using Stateflow charts as shown in Figure 3. It includes three concurrent state machines, namely, `SpineSchedulerEngine`, `SpinePktProcessingEngine`, and `Timer`.

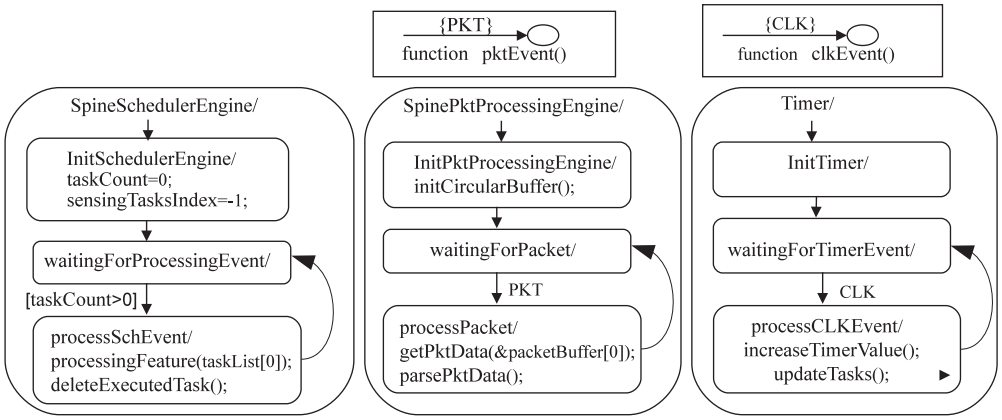



Fig. 3. Finite state machines for the SPINE node.

The `SpinePktProcessingEngine` state machine processes the incoming packets from the BS (represented in the chart as PKT events).

The `Timer` state machine keeps track of time and activates each task with the appropriate periodicity. It keeps a queue of task activation times and wakes up the `SpineSchedulerEngine` when a task needs to be executed.

The `SpineSchedulerEngine` executes the task to be run, for example, sampling and storing sensor data, computing signal processing functions, or sending data to the BS.

### 4.2. Code Generation

This flow uses the Stateflow Coder tool from The Mathworks and the HIFSuite from EDALab as code generators from the platform-independent model.

The ANSI C application code generated by The Mathworks Stateflow Coder is compiled by the toolchain of the ReISC processor. It is then simulated on the ISS for energy consumption analysis.

The SystemC behavioral model of the network node is generated from the Stateflow description using HIFSuite and the methodology presented in Lazarescu et al. [2012]. The SystemC module generated in this way directly matches the SCNSL interface model and can be used to create node instances in the network scenario.

The use of the generated code in simulations is presented in detail in Section 6.

## 5. HARDWARE MODELING

The hardware platform of the SPINE node consists of:

—the *ReISC* SoC that handles the sensors and performs data processing;
—the *RF Module* that handles the wireless communications.

### 5.1. ReISC System-on-Chip

The ReISC SoC was designed by STMicroelectronics and taped out at the end of 2009 using a 90nm technology. It is the first of a new family of ultra-low-power products.

The ReISC SoC includes the proprietary ReISC 3 core (Reduced-energy Instruction Set Computer). It provides hardware support for 8/16/20/32-bit data sizes, variable 16-bit-based instruction length, and secure data. ReISC 3 is targeted at ultra-low-power applications. It operates up to 50MHz and has 1MB FLASH and 32KB SRAM embedded memories and an extensive range of enhanced I/Os (one I2C, two GPIOs, two SPIs, one USART, and one USB) and peripherals (one 12-bit ADC, three general-purpose 16-bit

timers, and one internal timer). A comprehensive set of power-saving modes allows one to efficiently implement low-power applications.

For the optimization of the wearable SPINE node, the relevant peripherals are:

—the *Serial Peripheral Interface* (SPI), which handles the communication between the ReISC SoC and the RF Module;
—the *General Purpose Input/Output* (GPIO);
—the *timers*, which are used for synchronization purposes;
—the *Analog-to-Digital Converter* (ADC), which collects samples from the sensors;
—the *Reset and Clock Control Unit* (RCCU);
—the *Power Manager*, which is used to manage the power states of the ReISC SoC (CPU and peripherals).

The Serial Peripheral Interface (SPI) consists of a synchronous serial communication interface with a four-pin protocol. It allows half/full-duplex, synchronous, serial communication with external devices. There are separate buffers for reception and transmission, and the peripheral can operate in full-duplex mode. When the interface is configured as master, it provides the communication clock to the external slave device. The interface is also capable of operating in a multimaster configuration. It may be used for a variety of purposes, including simplex synchronous transfers on two lines, with an optional bidirectional data line, and reliable communication with CRC checking.

The General Purpose Input/Output (GPIO) is a set of pins whose behavior can be programmed by the application.

There are up to four timers in the ReISC Soc platform, which may be used for a variety of purposes, including measuring the pulse length of input signals (input capture) or generating output waveforms or counting events. The internal timer is the simplest one, and it can only count down.

The Analog-to-Digital Converter (ADC) converts the magnitude of an analog signal (voltage or current) into a digital representation with 12-bit resolution. The ReISC ADC provides 16 multiplexed channels, interrupt generation at the end of conversion, and DMA request generation during conversion.

The Reset and Clock Control Unit (RCCU) manages the power-on reset for the ReISC SoC system and generates the system clocks using PLLs. It also allows one to enable/disable the peripherals and their clocks.

The Dynamic Power Manager (DPM) activates clock gating for the unused peripherals, selects the power-down state for the analog hard macros when they are not required by the application, and selects the appropriate system clock frequency. In order to further reduce the power consumption, the DPM can also shut off the power domains for each ADC, for the PLL, and for the FLASH memory.

### 5.2. Power Consumption

The ReISC SoC implements dynamic power management techniques to reduce power consumption, such as clock gating and power gating, using two components: the Power Platform and the Dynamic Power Manager (DPM).

The Power Platform is a set of modules implementing local power states. To manage the power states of the CPU and the peripherals, the Power Platform uses three power islands. A power island is a region of the SoC with independent and scalable power supply as well as clock frequency. The organization of the power islands can be summarized as follows:

—An ALWAYS ON power island that includes the ReISC core, the RCCU, the Power Manager, Timers, and any other component that must be kept always enabled

Table I. SPI and ADC Peripherals' Power Consumption in Various Power Modes Relative to WORKING Mode

| Power Mode | SPI Power [%] | ADC Power [%] |
|---|---|---|
| OFF | 0 | 0 |
| NO CLOCK | 20 | N/A |
| IDLE | 40 | 20 |
| WORKING | 100 | 100 |

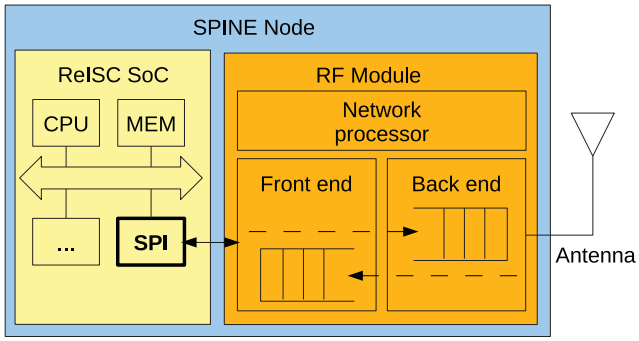



Fig. 4. Architecture of the RF Module and interconnection with the ReISC SoC.

—A FUNCTIONAL STATE power island (with flip-flop state retention) that includes the peripherals that can be switched on/off, for example, the SPI and the GPIOs
—An ANALOG power island that includes the ADC

The transitions between the power states can be controlled through an API of the DPM, which can be accessed at the application level. The DPM functions use device drivers to configure specific devices belonging to a power island. Finer control on power consumption can be obtained through the RCCU that allows one to selectively enable the peripherals and their clocks. Table I reports the power consumption of the SPI and ADC peripherals in various power modes, normalized with respect to their power consumption in the WORKING mode.

### 5.3. RF Module

The RF Module is connected through an SPI interface to the ReISC SoC in order to provide networking capabilities. The architecture of the RF Module inside the SPINE node is shown in Figure 4. It consists of three components: the front end, the network processor, and the back end.

The front-end component manages the SPI protocol to communicate with the ReISC SoC. It receives (1) data bytes from the ReISC SoC, to be sent to the network, and (2) packets from the network, to be sent to the ReISC SoC through the back end. The SPI transmission uses the simplex mode. The decision about setting the SPI controller in master or slave mode is made by the software running on the ReISC SoC. When the ReISC SoC SPI controller is in slave mode, it waits to receive data from the RF Module. When it enters the master mode, it cannot be interrupted by the RF Module. Therefore, the RF module uses a local FIFO queue to temporarily store the data to be transmitted to the ReISC SoC.

The network processor implements an IEEE 802.15.4 [LAN/MAN Standards Committee of the IEEE Computer Society 2006] medium access control (MAC). The back-end component addresses all the low-level details to send/receive data on the

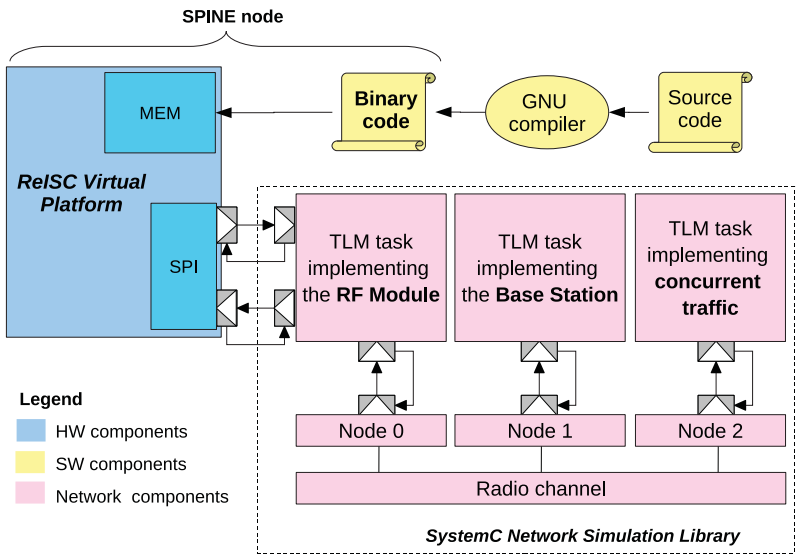



Fig. 5. Overview of the SystemC simulation setup and models.

radio channel. The transmission time of a packet is not constant, since the 802.15.4 MAC provides access based on random wait states with multiple retransmissions in case of failure. Therefore, the back-end component also contains a FIFO queue to store packets when the production rate of the ReISC SoC is higher than the transmission rate over the radio channel.

## 6. NETWORK MODELING

As shown in Figure 5, the network scenario consists of two parts: (1) the wireless channel and (2) several nodes accessing the channel. One of them is an accurate model of the wearable SPINE node that is implemented by connecting the ReISC SoC virtual platform, provided by STMicroelectronics, with the model of the RF Module. The other nodes are modeled at the behavioral level.

The ReISC virtual platform accurately models the hardware components of the ReISC SoC. The components closely coupled to the processor, such as the memory, are under the direct control of the ISS, while all other parts of the virtual platform (e.g., the peripherals) are modeled in SystemC. A SystemC wrapper also implements the interface between the ISS and the rest of the system.

The software components (i.e., the application and system code) are compiled as usual for the target platform. The binary is loaded into the memory module of the virtual platform and is executed by the ISS. The ISS is also used to drive at runtime the evaluation of a set of equations that model the power consumption of the node platform in its various power states. This allows one to achieve a cycle-accurate time profile of the power consumption for the whole node and for each part separately (processor and peripherals).

The ReISC virtual platform is connected to the network through a SystemC model of the RF Module.

The SCNSL classes are used to implement this scenario. In particular, the Task class is used for the functional models of the RF Module, the Base Station, and the sources of concurrent network traffic. One instance of the Node class is used for each node to

```
01  #include <systemc>
02  #include <tlm.h>
03  #include <scnsl.hh>
04  #include "Spine_t.hh"
05  #include "Base_Station_t.hh"
06  int sc_main( int argc, char * argv[] )
07  {
08    ...
09    Scnsl::Setup::Scnsl_t * scnsl = Scnsl::Setup::Scnsl_t::get_instance();
10
11    // Creation of nodes:
12    Scnsl::Core::Node_t * n0 = scnsl->createNode();
13    Scnsl::Core::Node_t * n1 = scnsl->createNode();
14
15    // Creation of the wireless channel:
16    ccs.channel_type = CoreChannelSetup_t::SHARED;
17    Scnsl::Core::Channel_if_t * ch = scnsl->createChannel( ccs );
18
19    // node-channel binding:
20    scnsl->bind( n0, ch, bsb0 );
21    scnsl->bind( n1, ch, bsb1 );
22
23    // Mapping of task instances on to nodes:
24    Base_Station_t t1( "BS", n0);
25    Spine_t t0( "SPINE0", n1);
26    ...
```

Fig. 6. The network scenario is described in SystemC using SCNSL primitives. Network elements are created as SystemC modules.

host its Tasks. Finally, the SCNSL Shared Channel is used to model the radio channel connecting all the Nodes.

As shown in Figure 6, the network is described in SCNSL by creating nodes, channels (with protocols), and tasks as standard SystemC modules (lines 12, 13, and 17). Lines 20–21 connect node instances to the shared channel, while lines 24–25 create task instances for the Base Station and a SPINE node.

The simulation effort (CPU time vs. simulated time) depends on the number of generated events per simulated time unit. In our approach, there are two sources of simulation events, namely, the ISS and the network simulation library (SCNSL). The ISS events depend on the software executed by the ReISC CPU, while the SCNSL events depend on the transmission and reception of packets among the nodes of the network. In the simulation scenarios considered in this article, the simulation effort is totally dominated by the ISS. Therefore, the scalability of the tool as a function of the number of network nodes mainly depends on the scalability of the SCNSL. The overhead added by the SCNSL simulation of the other network nodes, which are modeled at the behavioral level, is extremely low, as was shown, for example, in Crepaldi et al. [2013]. That paper proved that the approach is as efficient and scalable as other well-known network simulators like NS-2 [Fummi et al. 2008], as shown in Figure 7.

## 7. TESTING SCENARIOS

The simulation of the system is performed at the transaction level (TLM). Two TLM sockets connect the ReISC SoC virtual platform to the RF Module, and each SCNSL task is connected to the corresponding SCNSL node through an additional TLM socket.

The application is simulated by running the system code and the SPINE application under analysis (see Section 4.2) on the ISS of the ReISC SoC virtual platform. In this work, we focus on the minimization of the energy spent for computation, not for communication (which is thoroughly addressed in the literature). Therefore, we need very accurate power models of the ReISC SoC hardware and software components,

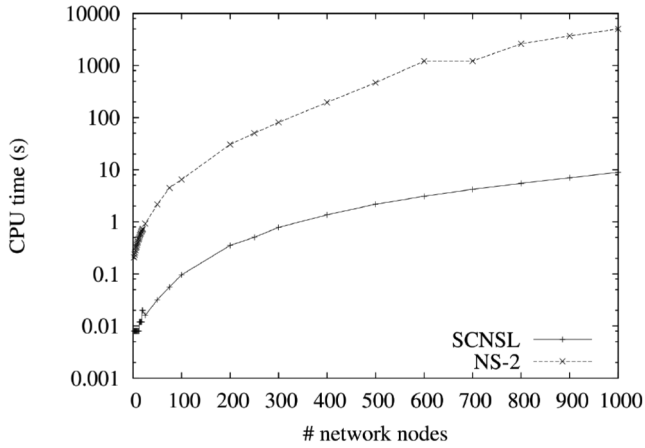



Fig. 7. Comparison of the simulation effort of SCNSL and NS-2 as a function of network size, using nodes modeled at the behavioral level.

while the power accuracy of the models of the transmission components is not our main focus.

The network setup consists of:

—one wearable SPINE node (described in Section 3.2);
—one Base Station, with two main functions:
(1) to initialize the wearable SPINE node using configuration packets;
(2) to listen for incoming data packets from the wearable SPINE node;
—a node to generate concurrent traffic on the shared channel, using the same communication protocol. The level of concurrent traffic from this node is varied to evaluate the effects of channel congestion on both the communication and the internal operation of the wearable SPINE node.

The nodes communicate with the BS in a star topology. They are bound to a shared channel model, which reproduces the behavior of the wireless medium in terms of packet loss and collisions.

The behavior of the wearable SPINE node is defined by the instructions executed by the ISS on the ReISC virtual platform. The other nodes in the network are modeled at the transaction level in SystemC, which also defines all network-related timings.

An algorithm to adapt the interpacket interval on the wearable nodes based on channel congestion level will be presented and analyzed in Section 7.3.

### 7.1. Application Accuracy

Figure 8 shows the fall detection accuracy of the BS algorithm as a function of the fall patterns and the sensor sampling rate. The detection accuracy was calculated as:

$$\text{fall detection accuracy} = \frac{\text{correct fall detections}}{\text{number of falls in the data set}} \times 100[\%]. \quad (1)$$

The undetected falls are much more important than the false positives for this application, since they mean undetected potential critical situations for the patients.

The detection accuracy was evaluated on several datasets that include between two and five fall patterns each. The results in Figure 8 show that the accuracy depends on both the fall pattern and the sensor sampling rate.

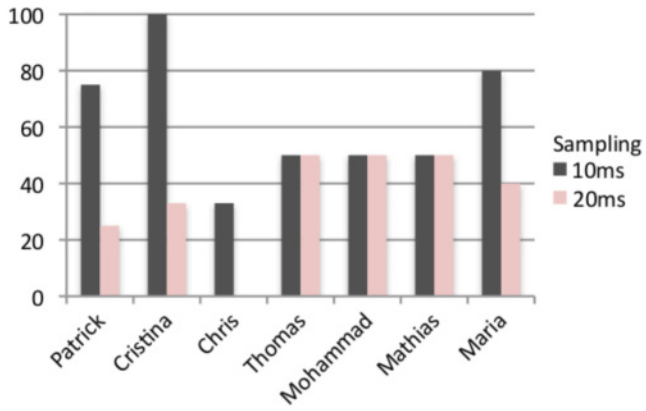



Fig. 8. The accuracy of the fall detection algorithm depends on the fall pattern and on the sampling rate of the sensor. Lower sampling rates reduce the fall detection accuracy.

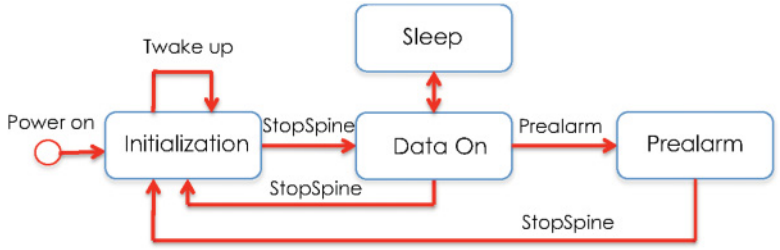



Fig. 9. SPINE state diagram.

The fall patterns depend on the body movements during the falls, which can vary with several physical and body parameters, such as body weight and fall height on the one hand and the initial position, movement, and speed on the other hand.

For a given pattern, the sensor sampling rate can influence the detection accuracy. Finer-grain sampling generally leads to higher fall detection accuracy. However, undersampling techniques can be used to reduce the traffic on temporarily congested communication channels, and its effects on the detection accuracy can be reduced using adaptable predictive algorithms [Mesin et al. 2014].

### 7.2. HW Performance

Figure 9 shows the power state diagram of the wearable ReISC SPINE node. In normal mode, the communication between the ReISC SoC node (running the SPINE application code generated from Stateflow as discussed in Section 4) and the SPINE BS node follows a well-defined sequence:

—`Initialization`: initialize the communication between the wearable ReISC SoC (SPINE) node and the BS.
—`DataOn`: send the sensor data from the wearable ReISC SoC (SPINE) node to the BS.
—`Sleep`: low-power state between samples or data transmissions.
—`Prealarm`: like `DataOn`, but disabling transitions to the `Sleep` state.

After power-on, the node enters the `Initialization` state. In this state, it wakes up every `Twake_up` interval (a parameter ranging from 10 seconds to 5 minutes) waiting for the BS to connect and send configuration packets (namely, three packets per feature).

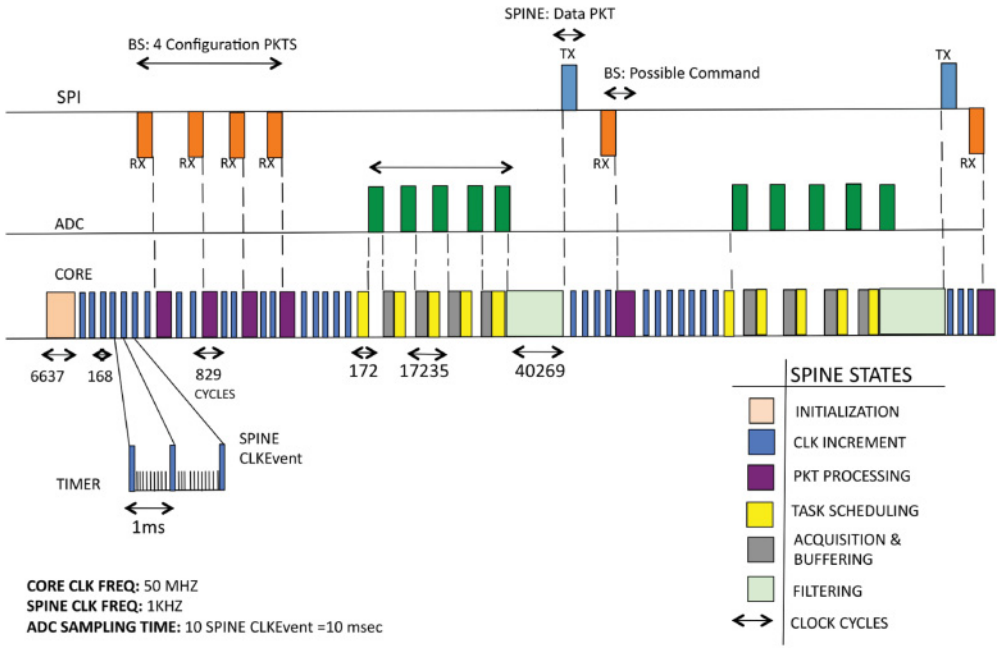



Fig. 10. Typical activation of the TIMER, SPI interface, ADC, and ReISC CORE elements of a wearable SPINE node.

The configuration ends with a `StartSpine` packet from the BS that requests the node to switch to the `DataOn` state.

In the `DataOn` state, the wearable SPINE node periodically collects samples and sends packets to the BS, according to the configured features. The node may go to the `Sleep` state between task scheduling times; this behavior is implemented by setting a timer. The wake–sleep duty cycle depends on a static configuration whose optimal setting will be determined by design space exploration.

The transition to the `Prealarm` state happens when a free-fall event is detected by the BS. In this case, the BS warns the wearable SPINE node using a `Prealarm` message. The wearable SPINE node switches off the timeout to the `Idle` mode and its behavior becomes similar to the `DataOn` state; that is, the node keeps sending data without going to sleep. At the same time, the BS also sends a prealarm message to the operator.

The return to the `Initialization` state of the wearable SPINE node is triggered by the `StopSpine` command.

In order to use application knowledge to exploit the idleness of the wearable node hardware components that are critical to the SPINE application, we consider the timing requirements of the application and use code profiling information to insert the appropriate power management function calls in the generated code. Thus, low-power and power-off strategies can be explored by simulating different power management settings.

For example, Figure 10 shows a typical activation sequence of the TIMER, SPI peripheral, ADC, and ReISC CORE elements. The SPI interface is used to exchange commands and data with the radio module. The node waits for four configuration packets from the BS (the burst of four RX transactions on the SPI activity line). After these, the node enters the normal operation mode, in which it periodically sends the sensor data (acquired and preprocessed on-board) and listens for acknowledgments

and commands from the BS (represented as pairs of TX and RX events on the SPI activity line).

On the CORE activity line in Figure 10, we can see how the TIMER wakes up the SPINE scheduler every 1ms to poll for tasks that need to be run. External events, like a packet received from the radio module over the SPI interface, trigger the activation of the appropriate SPINE task for their processing.

After the initial configuration packets, the scheduler activates periodically the task that performs the sensor reading. Each reading consists of five raw samples acquired by the ADC and their processing by the filtering task. The output of the filter is sent to the radio module using a TX packet over the SPI interface. The radio module will then autonomously transmit the packet to the BS. Right after the transmission, the radio module will listen for a brief time for BS acknowledgment and possibly for commands.

Most node processing consists of sensor sample acquisition, filtering, and communication to the radio module. Thus, we will focus on these activities to optimize the power consumption of the node.

### 7.3. Impact of Communications

In the following, we describe the wearable node operation and its network interaction.

After the initialization phase, the wearable SPINE node listens for 16-byte configuration packets from the BS, which are sent at 100ms intervals. The radio module sends each received packet to the SPI controller that is configured as slave and raises an interrupt. The SPI interrupt handler (ISR) copies the packet payload to the SPINE packet buffer, from where the application can retrieve it to configure the node accordingly. The last configuration packet from the BS contains the `StartSpine` command. This starts the SPINE engine, sets the SPI controller to master mode, and initializes the timer to the packet transmission interval configured by the BS.

The ADC is then configured to sample the acceleration data from the accelerometer sensor over three channels. After every sample is read, the ADC interrupt is raised and its ISR places the value in the SPINE circular buffer. When a full sample set is ready, it is processed, filtered, and sent to the RF Module through the SPI controller.

Note that the accelerometer reading and the data packet rates are independent. The former can be higher to prevent detection accuracy degradation and save radio power.

We simulated 2 seconds of system time with the following settings:

—From 0 second to 0.4 second, the BS sends four configuration packets.
—From 0.4 second to 2.0 seconds, the wearable SPINE node sends sensor data packets to the BS.
—From 1.0 second to 2.0 seconds, concurrent traffic saturates the communication channel. This emulates the worst-case networking conditions that can happen due to radio traffic from several unrelated nodes.

The simulation took about 590 seconds on a workstation with a 3GHz Intel processor. As we discussed in Section 6, this time is dominated by the ISS execution, while the simulation performance impact of the nodes modeled using SCNSL is extremely low. Thus, to reduce the simulation effort, the ISS is used only for the nodes for which instruction-level accuracy is needed. In our experiments, one SPINE node is simulated by the ISS, while the BS node and up to five concurrent nodes are simulated at the behavioral level by SCNSL.

The simulation results in Figure 11 show the time spent by the wearable SPINE node to send a packet to the BS. This varies from 1.9ms to 42ms based on the channel congestion level, with peaks during congestion that are due to the accumulation of the backoff times of the protocol for retransmission attempts in case of collisions.

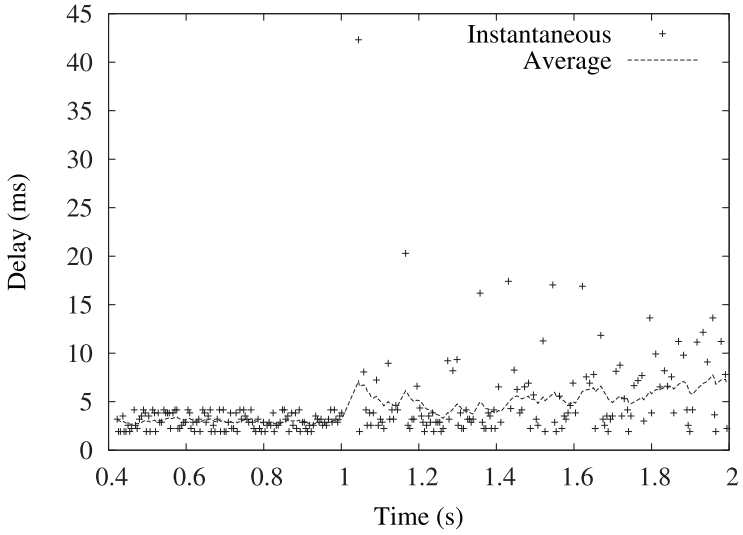



Fig. 11. Instantaneous and average time spent by the wearable SPINE node to send a packet to the BS for various channel conditions: 0.4–1.0 second free channel, 1.0–2.0 seconds congested channel.

Moreover, in this test case, the transmission delay also causes the loss of 76 packets during the periods of congestion. The packets can be lost when the actual transmission rate of the RF Module falls below the packet production rate of the ReISC SoC and the transmission queue of the RF Module fills up.

This result suggests that the optimal output rate may be determined in some cases by the channel conditions and may fall below the application requirements. Thus, knowledge of the channel status can be used to improve the scheduling of the activities of the wearable SPINE node to reduce its energy consumption. Assuming that the application requirements can accommodate a range of network transmission delays, we examine in the following how adaptive packet transmission can reduce both the channel contention and the overall node energy consumption. Please note that no realistic application scenario requires subsecond network latencies in this case study. Thus, the proposed strategy can effectively improve battery lifetime without compromising functionality.

The experiment shows that the quality of the fall detection is gradually affected by the transmission delays (see Section 3.1). This means that, based on application quality-of-service requirements, we may use channel congestion statistics to reduce the wearable SPINE node activity and improve its energy consumption. For this, we need both a way to communicate the network statistics from the RF Module to the ReISC SoC and an application-level algorithm to adapt the node activities accordingly.

The former can be implemented using the SPI channel that connects the RF Module to the ReISC SoC on the node. The SPI transfers radio data in both directions, as well as configurations for the RF Module and channel quality statistics computed by the radio module.

The data exchanged over the SPI channel are organized in messages structured as shown in Figure 12. The first message (Figure 12(a)) is sent by the application to configure the 802.15.4 parameters of the RF Module: node role (coordinator, router, end device), its 64-bit MAC address, network identifier, acknowledgment of transmission, and its behavior when it is not transmitting (stay idle or listen). The second message (Figure 12(b)) is used to transfer packets to or from the network. The third message

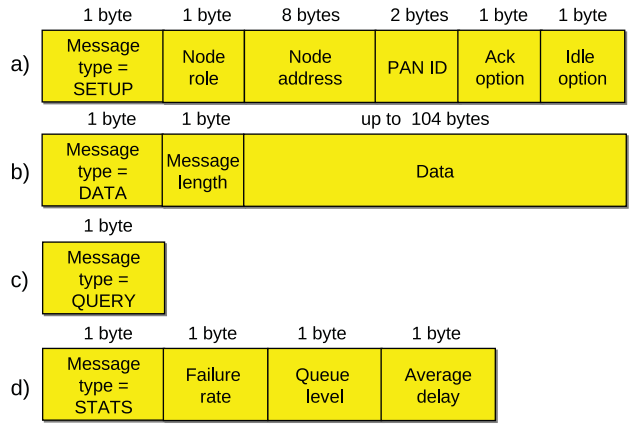



Fig. 12. Structure of the SPI messages exchanged between the ReISC SoC and the RF Module: configuration command (a), data message (b), statistics request (c), and statistics response (d).

Table II. Packet Transmission Performance and Relative Energy Consumption of the SPI and ADC Peripherals as a Function of the Transmission Policy, Normalized to a Nonadaptive Policy

| Transmission Policy | SPI Transfer Attempts | BS-Received Packets | Packet Loss Ratio [%] | Peripheral Energy [%] |
|---|---|---|---|---|
| Nonadaptive | 332 | 256 | 23 | 100 |
| $K = 1$ | 300 | 233 | 22 | 92 |
| $K = 4$ | 216 | 189 | 12 | 74 |
| $K = 5$ | 205 | 182 | 11 | 72 |
| $K = 6$ | 186 | 173 | 7 | 68 |

(Figure 12(c)) is sent by the application to ask for packet transmission statistics. The fourth message (Figure 12(d)) is the RF Module reply to this query, reporting the number of transmission failures over 255 requests, the number of packets in the output queue, and the average delay to complete a transmission (the metric shown in Figure 11). For all messages, the first byte encodes the type of the message.

Packet transmission failures happen either when the maximum number of transmission attempts is reached because of a busy channel or when the maximum number of retransmissions is reached without receiving an acknowledgment. Both statistics can be used to estimate the channel conditions and to adapt the packet transmission delay (`Delay`) to those conditions as follows:

(1) Initialize `Delay` to 1ms.
(2) If the last SPI transmission failed due to buffer overflow, then set $\texttt{Delay} = \texttt{Delay} \times 2^K$ (up to a maximum allowed by the application requirements).
(3) If the last SPI transmission was successful, then set $\texttt{Delay} = \texttt{Delay}/2$.

Figure 13 shows the packet transmission delay in different channel conditions, calculated using the algorithm earlier with $K = 1$. The plot shows that during channel congestion, the wearable SPINE node sends data less frequently.

Table II reports the transmission performance and the total energy consumption of the SPI and ADC peripherals as a function of the transmission policy, normalized to the nonadaptive policy value. The attempts to transfer data to the RF Module through the SPI are a measure of the application activity acquiring and processing data. The number of packets received by the BS measures the amount of useful data

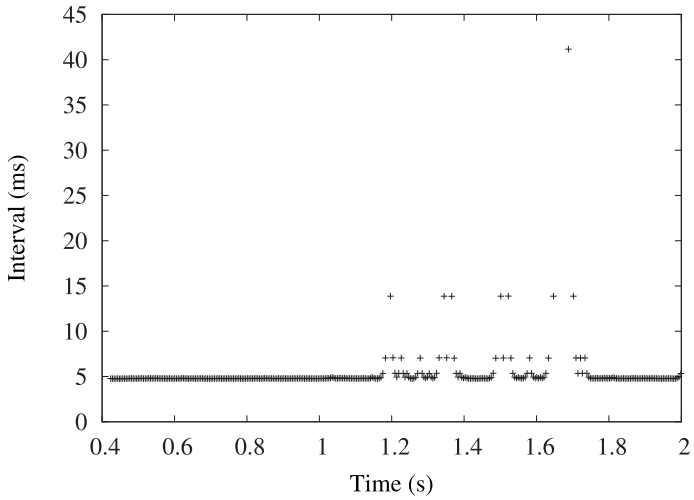



Fig. 13. Time interval between two SPI data messages as a function of simulation time for various channel conditions: 0.4–1.0 second free channel, 1.0–2.0 seconds congested channel.

that reached the destination. The packet loss ratio represents the percentage of data that was acquired and processed but lost in the transfer to BS. The relative energy consumption is calculated using the total energy needed by the ADC and SPI peripherals, whose activity is modulated based on channel congestion.

The results show that a more aggressive adaptation to channel congestion level (higher $K$) reduces both the packet loss (thus increasing transmission reliability) and the energy consumption, since the ADC and SPI are active for a fixed time for each new sensor acquisition.

## 8. DESIGN SPACE EXPLORATION

In this section, we show how the infrastructure presented in this article allows one to perform a complete analysis of the system to optimize its power consumption using the DPM features, by taking into consideration both the network conditions and the application requirements.

### 8.1. Setup

Once the application has been mapped to the HW platform, the analysis and choice of which system configuration is most suitable requires an iterative evaluation phase.

In this work, this exploration phase is automated using the analysis capabilities of *MOST—Multi-Objective System Tuner* [Palermo et al. 2009], a design space exploration tool supporting platform-based design. The goal of the tool is to select at design time the most promising configurations in terms of power consumption, performance, and quality of service for both the hardware platform and the application running on it. It provides easy exploration control features and data postprocessing capabilities.

The exploration loop that we used automatically configures the simulation platform at the node, network, and application levels. The parameterization of the system under analysis can be summarized as follows:

—*Parameter K* of the adaptive transmission policy that is analyzed in Section 7, that is, {0 (for non-adaptive), 1, 4, 6, 8}.

—*Power policies* for the management of the hardware platform Voltage Islands (VIs) (ADC = {ON, OFF}, SPI = {ON, NO-CLOCK, SNOOZE[2]}). The *ON* value means that the VI is always on, thus alternating its power consumption between the *IDLE* and *WORKING* power states (see Table I) depending on the workload. Other power state values (which are dependent on the VI type) can be adopted instead of *IDLE* to save power when the corresponding units are not used. The change of the power mode is done when a long period of inactivity is expected (e.g., as shown in Figure 10) by calling the suitable routines directly in the application code. The power policy is kept constant during the evaluation period.
—*Number of concurrent nodes* excluding the BS (`ConcurrentNodes`). This parameter controls the number of active nodes that produce network traffic at the same time, that is, {1, 3, 5}.

We are interested in evaluating the following metrics:

—*Packet loss rate*, calculated as the ratio between the number of lost packets and the total number of packets that were sent
—*Average transmission interval*, calculated as the average time between two packets received by the Base Station
—*Total energy*, calculated as the energy consumed to send a given number of packets[3]

These metrics were selected to allow the monitoring of the behavior of the system energy and the quality of service while varying the network conditions (i.e., the number of concurrent transmissions) and the application-level tunable parameters (i.e., the transmission adaptivity factor and the power policies for the voltage islands).

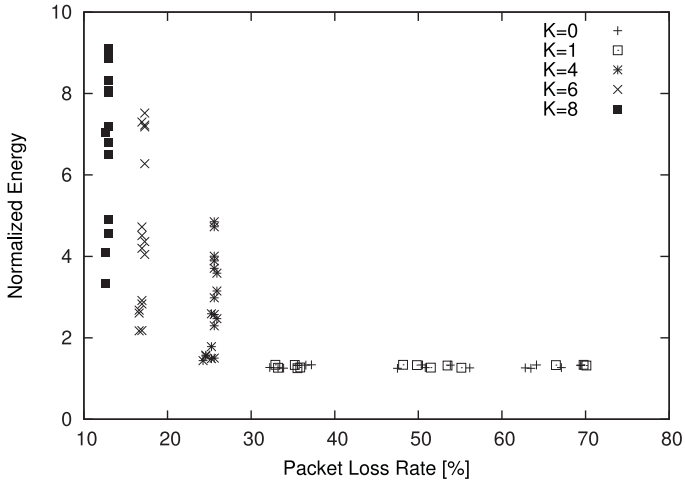
### 8.2. Results

Figure 14 shows the scatter plots of the design configurations explored in the *Total Energy–Packet Loss* space. We highlighted the configurations with different *Adaptivity Factor K* (Figure 14(a)) and *Number of Concurrent Nodes* (Figure 14(b)). A similar scatter plot for the *Average Reception Interval–Packet Loss* space is shown in Figure 15.

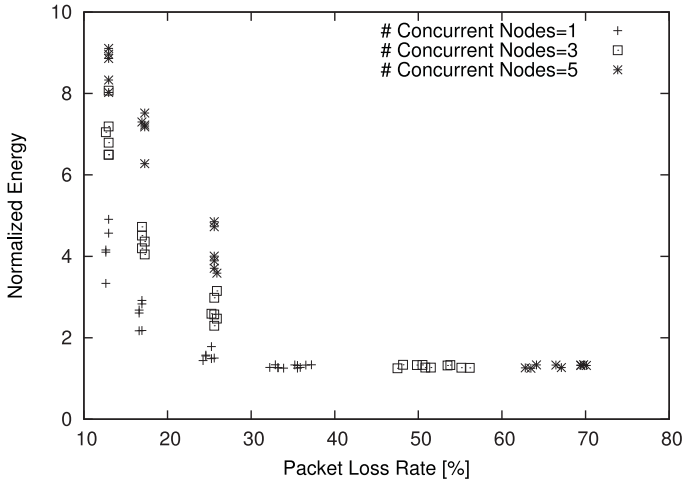
Figure 14(a) shows that the system behavior can be divided into two areas, depending on the adaptivity values. For $K < 4$, the packet loss rate is very high and the behavior of the configurations with $K = 0$ and $K = 1$ mostly overlap. The design points appear clustered by packet loss rate depending on network size (see Figure 14(b)). Additionally, the energy consumed by the system is almost constant, due to the low activity rates of the computing and sensing parts of the node, since the most energy-intensive part is the communication module. $K \geq 4$ means higher adaptivity, which makes the application more tolerant to network size changes (see Figure 14(b)). This enables a more robust behavior, that is, an almost constant rate of packet loss rate. However, the energy consumption increases with the increase of the number of concurrent nodes due to the higher message overhead for channel arbitration. Additionally, the energy values of the design configurations for the same number of concurrent nodes and adaptivity values vary also with the power policies.

A similar analysis can be done for Figure 15, which represents the average transmission interval as a function of packet loss rate. For $K < 4$ (less adaptivity), Figure 15 shows that higher values of packet loss impact the average transmission interval due

[2]The *SNOOZE* power state of the SPI VI is like the *OFF* power state of the ADC VI, but it retains the flip-flop states.
[3]We will present the normalized values of the *total energy* with respect to the lowest energy consumption value across the entire design space in order to allow a more abstract evaluation of the different design alternatives.




(a)



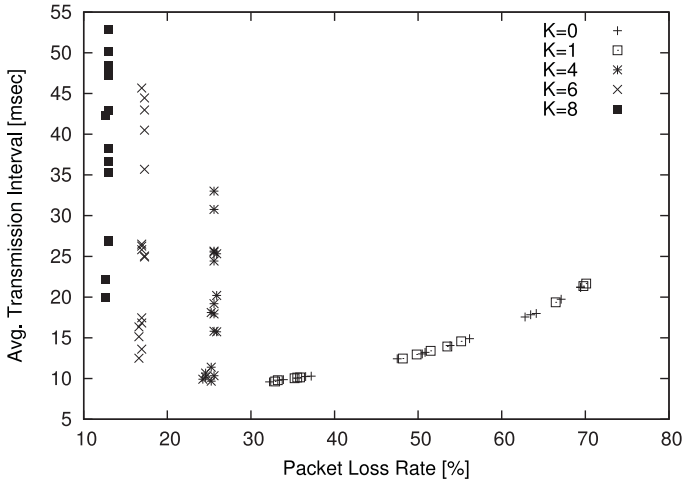

(b)

Fig. 14. Scatter plot in the *Total Energy–Packet Loss* space by varying the *Adaptivity Factor K* (a) and the *Number of Concurrent Nodes* (b). The energy is normalized to the lowest consumption.

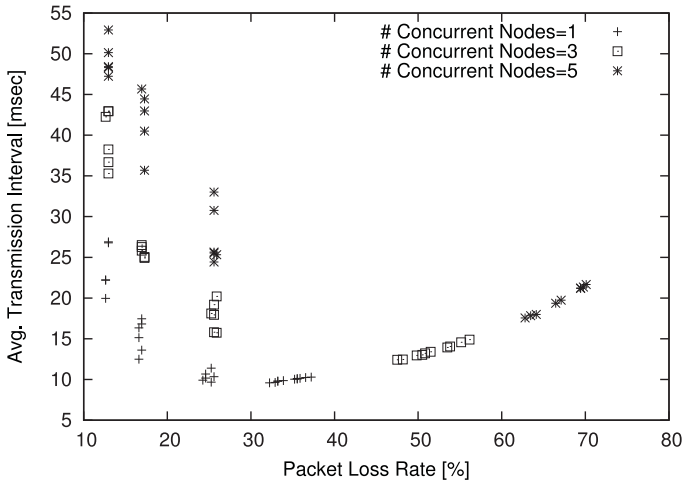
to retransmissions caused by a larger number of active nodes (see Figure 15(b)). However, the increased adaptivity for $K \geq 4$ reduces the packet loss rate, but at the cost of higher transmission intervals. This is a tradeoff that requires a more detailed designer analysis, since the best value may also depend on the application requirements.

Figure 16 shows a more detailed analysis of the energy metric as a function of the power policies and the active network size. This analysis is presented using energy contour maps where the values at each point (in terms of power policies and network size) have been averaged across all other parameters.

Figure 16(a) shows the energy contour map by varying the ADC and SPI power policies. As expected, by activating the power policies for both voltage islands (ADC VI-*OFF* and SPI VI-*SNOOZE*, upper-right corner), the energy is reduced by 15% on average with respect to the base case (bottom-left corner). However, the same figure




(a)




(b)

Fig. 15. Scatter plot in the *Average Reception Interval–Packet Loss* space by varying the *Adaptivity Factor K* (a) and the *Number of Concurrent Nodes* (b).

shows that the configuration ADC VI-*OFF* and SPI VI-*NO-CLOCK* consumes more energy than the configuration where one of the two islands is kept always *ON*. This is mostly due to the combined overhead of the power state transitions in terms of both latency and energy. In fact, the power savings of the SPI VI-*NO-CLOCK* periods are wasted by the overheads associated with power state transitions. As mentioned earlier, the phenomenon does not occur in the SPI VI-*SNOOZE* case, since the power reduction achieved in this state is much higher than the transition overheads.

These effects can be analyzed in more detail in Figure 16(b), which shows the energy contour map by varying the SPI power policy and the active network size. The energy impact of the SPI power policies is not obvious in that figure, since it interacts with the active network size. In fact, the phenomenon in Figure 16(a), which shows that it is more energy efficient to keep the SPI VI-*ON* than to switch it to *NO-CLOCK*, occurs

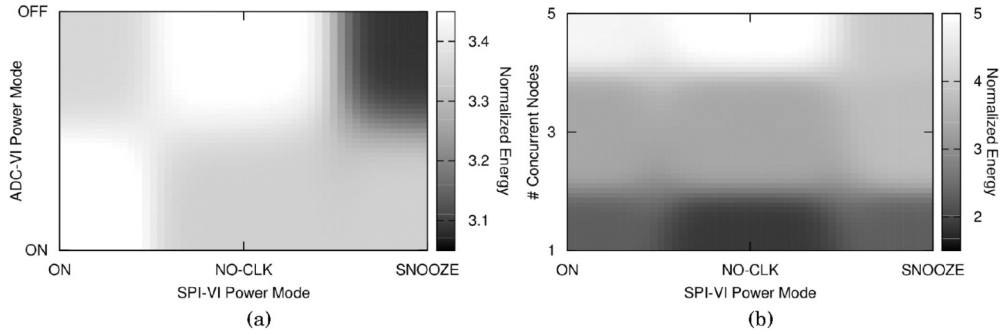



Fig. 16. Energy contour map by varying the *Power Policies* for both SPI and ADC voltage islands (a) and the SPI *Power Policy* and the *Number of Concurrent Nodes* (b). The energy is normalized to the lowest consumption over the entire design space.

only when the number of concurrent nodes is equal to five (or, presumably, larger). On the other hand, with three concurrent nodes, the *NO-CLOCK* power policy seems to have no effect (either positive or negative), while for a network with only one concurrent node, the *NO-CLOCK* state produces the largest energy gain.

The various energy behaviors analyzed in this design space exploration allow one to implement and optimize an ADC and SPI power policy that adapts to the active network size to effectively save node energy.

## 9. CONCLUSION

This article addresses the need to provide an effective design space exploration flow to optimize the power consumption of distributed embedded applications running on modern heterogeneous SoC devices.

The proposed flow allows one to efficiently verify the behavior and explore the optimization solutions for networked embedded systems. It uses model-driven design, automated code generation, and system-level simulation, enabling an effective design space exploration starting from the early design stages.

The design entry is facilitated by providing a graphical environment for platform-independent application modeling and simulation based on concurrent Stateflow StateCharts. They can be automatically converted, using specific generators, into SystemC platform-independent architectural models and platform-dependent application C code.

Unlike many existing network and node simulators, SystemC can perform the holistic system simulations needed for an efficient exploration of multiple network- and application-aware power management parameters and strategies. An Instruction Set Simulator and a cycle-accurate power model of the hardware platform were included in the SystemC system-level simulation to allow a detailed analysis of the SoC behavior and of its power consumption while running the application in a realistic network scenario.

The experimental results that we obtained using this flow demonstrate its effectiveness on a representative use case. We are able to explore several system configurations in realistic operation conditions that provide the developer the means to choose the best configuration, based on the application requirements, and without expensive and time-consuming field trials.

The proposed flow can be used for the analysis and optimization of a broad range of distributed embedded applications.

## REFERENCES


X. Chang. 1999. Network simulations with OPNET. In *Proceedings of the 31st Conference on Winter Simulation: Simulation—A Bridge to the Future - (WSC'99)*, P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans (Eds.), Vol. 1. ACM, New York, NY, USA, 307–314. DOI=10.1145/324138.324232 http://doi.acm.org/10.1145/324138.324232

E. Cheong, E. A. Lee, and Y. Zhao. 2005. Viptos: A graphical development and simulation environment for TinyOS-based wireless sensor networks. In *SenSys*, Vol. 5. ACM, New York, NY, 302–302.

M. Crepaldi, P. M. Ros, D. Demarchi, J. Buckley, B. O'Flynn, and D. Quaglia. 2013. A physical-aware abstraction flow for efficient design-space exploration of a wireless body area network application. In *Euromicro Conference on Digital System Design (DSD'13)*. 1005–1012. DOI: http://dx.doi.org/10.1109/DSD.2013.114

S. Croce, F. Marcelloni, and M. Vecchio. 2008. Reducing power consumption in wireless sensor networks using a novel approach to data aggregation. *In The Computer Journal* 51, 2 (2008), 227–239. DOI: 10.1093/comjnl/bxm046 http://comjnl.oxfordjournals.org/content/51/2/227

D. Quaglia and F. Stefanni 2013. SystemC Network Simulation Library – version 2. Retrieved from http://sourceforge.net/projects/scnsl.

W. Du, F. Mieyeville, D. Navarro, and I. O. Connor. 2011. IDEA1: A validated SystemC-based system-level design and simulation environment for wireless sensor networks. *EURASIP Journal on Wireless Communications and Networking* 2011, 1 (Oct. 2011), 1–20. DOI: http://dx.doi.org/10.1186/1687-1499-2011-143

W. Du, D. Navarro, F. Mieyeville, and F. Gaffiot. 2010. Towards a taxonomy of simulation tools for wireless sensor networks. In *3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools'10)*, Vol. 9. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, Belgium, Article 52, 7 pages. DOI: http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8659

F. Fummi, D. Quaglia, and F. Stefanni. 2008. A SystemC-based framework for modeling and simulation of networked embedded systems. In *Proceedings of Forum on Specification & Design Languages*. 49–54. DOI: http://dx.doi.org/10.1109/FDL.2008.4641420

L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin. 2004. EmStar: A software environment for developing and deploying wireless sensor networks. In *USENIX Annual Technical Conference, General Track*. Advanced Computing Systems Association, Berkeley, CA, 283–296.

R. Gravina, A. Guerrieri, G. Fortino, F. Bellifemine, R. Giannantonio, and M. Sgroi. 2008. Development of body sensor network applications using SPINE. In *Systems, Man and Cybernetics*. IEEE, 2810–2815. DOI: http://dx.doi.org/10.1109/ICSMC.2008.4811722

K. Grüttner, P. A. Hartmann, K. Hylla, S. Rosinger, W. Nebel, F. Herrera, E. Villar, C. Brandolese, W. Fornaciari, G. Palermo, C. Ykman-Couvreur, D. Quaglia, F. Ferrero, and R. l Valencia. 2013. The COMPLEX reference framework for HW/SW co-design and power management supporting platform-based design-space exploration. *Microprocessors and Microsystems* 37, 8, Part C (2013), 966–980. DOI: http://dx.doi.org/10.1016/j.micpro.2013.09.001 Special Issue on European Projects in Embedded System Design: {EPESD2012}.

LAN/MAN Standards Committee of the IEEE Computer Society. 2006. *IEEE Standard for Information Technology — Telecommunications and Information Exchange Between Systems — Local and Metropolitan Area Networks — Specific Requirements — Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (LR-WPANs)*. Technical Report. IEEE.

M. Lazarescu, P. Sayyah, D. Quaglia, and F. Stefanni. 2012. SystemC model generation for realistic simulation of networked embedded systems. In *Digital System Design (DSD)*. IEEE, 423–426.

P. Levis, N. Lee, M. Welsh, and D. Culler. 2003. TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*. ACM, New York, NY, 126–137.

S. Liao, G. Martin, S. Swan, and T. Grötker. 2002. *System Design with SystemC*. Kluwer Academic Publishers, Dordrecht, the Netherlands.

M. Lora, R. Muradore, R. Reffato, and F. Fummi. 2014. Simulation alternatives for modeling networked cyber-physical systems. In *Euromicro Conference on Digital System Design (DSD'14)*. 262–269.

F. Mattern and C. Floerkemeier. 2010. From active data management to event-based systems and more. In the *Internet of Computers to the Internet of Things*. Springer-Verlag, Berlin, 242–259.

S. McCanne, S. Floyd, K. Fall, and K. Varadhan. 1989. Network Simulator NS-2. (1989). http://www.isi.edu/nsnam/ns.

L. Mesin, S. Aram, and E. Pasero. 2014. A neural data-driven algorithm for smart sampling in wireless sensor networks. *EURASIP Journal on Wireless Communications and Networking* 2014, 1 (2014), 23. DOI:http://dx.doi.org/10.1186/1687-1499-2014-23

J. A. Miller, R. S. Nair, Z. Zhang, and H. Zhao. 1997. JSIM: A Java-based simulation and animation environment. In *Simulation Symposium, 1997*. IEEE, 31–42. DOI:http://dx.doi.org/10.1109/SIMSYM.1997.586473

M. M. R. Mozumdar, L. Lavagno, L. Vanzago, and A. L. Sangiovanni-Vincentelli. 2010. HILAC: A framework for hardware in the loop simulation and multi-platform automatic code generation of WSN applications. In *International Symposium on Industrial Embedded Systems (SIES)*. 88–97. DOI:http://dx.doi.org/10.1109/SIES.2010.5551370

F. Mulas, A. Acquaviva, S. Carta, G. Fenu, D. Quaglia, and F. Fummi. 2010. Network-adaptive management of computation energy in wireless sensor networks. In *ACM Symposium on Applied Computing (SAC) (SAC'10)*. ACM, New York, NY, 756–763.

F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. 2006. Cross-level sensor network simulation with COOJA. In *Local Computer Networks*. IEEE, 641–648. DOI:http://dx.doi.org/10.1109/LCN.2006.322172

G. Palermo, C. Silvano, and V. Zaccaria. 2009. ReSPIR: A response surface-based Pareto iterative refinement for application-specific design space exploration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28, 12 (2009), 1816–1829. DOI:http://dx.doi.org/10.1109/TCAD.2009.2028681

J. Polley, D. Blazakis, J. McGee, D. Rusk, and J. S. Baras. 2004. ATEMU: A fine-grained sensor network simulator. In *Sensor and Ad Hoc Communications and Networks*. IEEE, 145–152.

Gyula Simon, Peter Volgyesi, Miklós Maróti, and Ákos Lédeczi. 2003. Simulation-based optimization of communication protocols for large-scale wireless sensor networks. In *IEEE Aerospace Conference*, Vol. 3. IEEE.

M. Streubühr, R. Rosales, R. Hasholzner, C. Haubelt, and J. Teich. 2011. ESL power and performance estimation for heterogeneous MPSOCS using SystemC. In *ECSI Forum on Specification and Design Languages*. 1–8.

The Mathworks. 1998. *MATLAB User's Guide*. Technical Report. The Mathworks. Retrieved from http://www.mathworks.com/.

B. L. Titzer, D. K. Lee, and J. Palsberg. 2005. Avrora: Scalable sensor network simulation with precise timing. In *Information Processing in Sensor Networks*. IEEE, 477–482.

A. Varga and R. Hornig. 2008. An overview of the OMNeT++ simulation environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, networks and Systems & Workshops (Simutools'08)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, Belgium, Article 60, 10 pages.