Simulation of Hardware and Software in Heterogeneous Wireless Sensor Network

David Navarro, Fabien Mieyeville, Mihai Galos, and Laurent Carrel
Université de Lyon, Institut des Nanotechnologies de Lyon (INL)
UMR5270 - CNRS, Ecole Centrale de Lyon, Ecully, F-69134, France
David.Navarro@ec-lyon.fr, Fabien.Mieyeville@ec-lyon.fr, Mihai.Galos@ec-lyon.fr, Laurent.Carrel@ec-lyon.fr

Abstract – **This paper presents a new feature of the IDEA1 Wireless Sensor Network (WSN) simulation platform: its ability to run simulations on heterogeneous sensor nodes that compose a network. This platform allows system-level simulations with hardware accurate models, with graphical inputs and outputs to easily simulate such distributed systems. When comparing IDEA1 simulation results to physical measurements, difference is 6 percent. IDEA1 is more than three times faster simulation compared to another simulator (NS2). In the testbed we consider, the well-known IEEE 802.15.4 standard is considered and different microcontroller units (MCU) and radiofrequency units (transceivers) compose the heterogeneous nodes. Output curves, packet delivery rate (PDR), packet latency can be evaluated. Moreover, energy consumption of sensor nodes is detailed with a very fine granularity, at hardware and software level. Indeed, energy consumption of each internal block of each device on each node can be monitored with IDEA1. Therefore, it is possible to simulate quickly and accurately heterogeneous (hardware different) nodes. Indeed, multitude of hardware platforms and communication standards lead to inter-communicating heterogeneous networks. This simulation platform can be used to explore design space in order to find the hardware devices and IEEE 802.15.4 algorithm that best fit a given application with a constrained energy budget.**

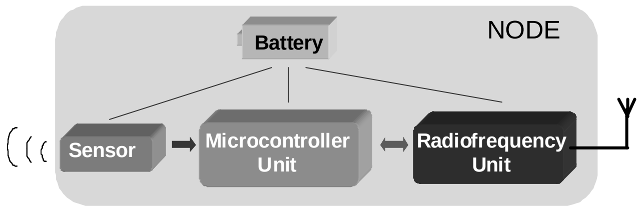
Keywords – Wireless Sensor Network; WSN; heterogeneous; simulation; model; SystemC.

I. INTRODUCTION

Wireless Sensor Networks (WSN) are widespread sensor systems. This paper presents IDEA1, a Wireless Sensor Networks simulator, as briefly presented in The Sixth International Conference on Sensor Technologies and Applications (SENSORCOMM) [1]. Wireless Sensor Networks are used in a large variety of applications, such as environmental data collection, security monitoring, logistics or health [2]. Wireless Sensor Networks are composed of resource-constrained sensor nodes that are deployed at different locations. The sensor nodes cooperatively monitor physical or environmental conditions, such as temperature, sound, vibration or pressure. Because of autonomy requirements, they have a specific architecture; they are typically composed of one or more sensors, an 8-bit or 16-bit microcontroller, sometimes an external non-volatile memory, a radiofrequency unit (transceiver) and an energy supply. Limited resources are energy, memory and processing capabilities. As mentioned in [3], many different platforms exist, and hardware heterogeneity is now a reality. Indeed, standards like IEEE 802.15.4 permit heterogeneous nodes to communicate. Meanwhile, such networks have to be simulated in order to estimate performances of the network.

The typical hardware architecture of a sensor node is detailed in Fig. 1. As introduced previously, it is composed of a sensor, a microcontroller, a radiofrequency unit (transceiver) and a battery. The sensor converts physical data into electrical signal. Microcontroller is the central element in the node as it executes user software. It embeds an analog to digital converter that is connected to sensor, a synchronous serial communication block that is connected to radiofrequency unit, power aware functions (like sleep or power-down modes). Radiofrequency unit give the possibility of remote connections to other nodes and gives the wireless functionality in the network. A battery supplies all the circuits; its characteristics give node autonomy.

Manufacturers of WSN hardware include ATMEL, Texas Instruments or Microchip microcontrollers and Texas Instruments, ATMEL, Freescale, or ST-Micro-electronics radiofrequency units. Many manufacturers supply sensors and battery modules. WSN applications are mainly low data rate.



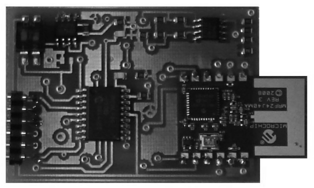


Figure 1. Typical wireless sensor node architecture, block diagram and hardware example N@L

For high data rate applications and intensive computations, Linux systems composed of 32-bit RISC processors are preferred but energy consumption is still prohibitive and autonomy is largely affected. Examples are the well-known Crossbow's Stargate platform [4] (Intel X-Scale processor at 400 MHz), or the TI CC2538 [5]. Even if these architectures will be probably more and more used in the future even though these systems shall be low-powered, they are for the moment relegated to the border of the WSN field. As stated in [3], 8-bit and 16-bit architectures represent 75% of microcontrollers in WSN applications. We also do not consider high data rate systems for the moment, and we focus on several months of battery life systems. Meanwhile, our platform is able to support these circuits if new models are included in the framework.

Wireless Sensor Networks design is a difficult task because designers have to develop a network at system level, with low-level (at sensor node: hardware and software) constraints. Therefore, CAD tools are required to make system-level simulations, considering low-level parameters. Our simulator IDEA1 permits that. Thus, we detail in this paper a new feature of our simulation platform: heterogeneous sensor nodes support.

Structure of the paper is as follows. Section II gives state of art on WSN simulators and basis of our work. Section III details our work: models and IDEA1 simulator. Section IV presents classical simulation results and related work. Section V shows latest results on heterogeneous simulation results.

II. Wireless sensor networks simulators

Many simulators were developed over the last few years. Most of them are restricted to specific hardware or focus on either network level or node level. Research on sensor network evaluation can be broadly divided in two categories: network simulators enhanced with node models, and node simulators enhanced with network models. A more detailed description is available in [6]. A summary and the heterogeneity support are detailed in Table I.

Typical network simulators are general-purpose network simulators, such as Network Simulator 2 (NS2) [7] and OMNeT++ [8] (and their declinations).

NS2 [7], an event-driven object-oriented network simulator belonging to NSNM, is by far the most used simulator [9] in the Mobile Ad hoc NETworks (MANETs) domain. Simulations are implemented in the C++ language and Object-oriented Tcl (OTcl). Protocols and extension libraries are written in C++; creation, control and management of simulations in OTcl. The extension policy of NS2 library has greatly contributed to its popularity, many protocols being implemented by the scientific community. WSN-specific protocols were implemented in NS2 among which a version of the IEEE802.15.4 standard. Large-size networks are difficult to implement because of their memory requirements and their simulation time [10]. Furthermore, detailed energy models for the different hardware and software elements of the node are lacking, resulting in poor precision at high abstraction level. Among the extensions of NS2 dedicated to WSN, SensorSim was developed too. Criticisms often made to NS2 are about the interdependences between modules resulting from its object-oriented structure. Hence, developing protocols for the NS2 library is complex and requires from developers a thorough knowledge of the software architecture of NS2. In the network community where standard protocols are clearly identified, such a limitation can be tolerated, but in WSN field, where no real standard was adopted and where research in protocols domain remains dominant, these mixing-up of modules become a hindrance to WSN-specific library development. Indeed, even if IEEE 802.15.4 or Zigbee are widespread, the increasing need for always-lower power consumption keeps the protocols domain in the most active research field in WSN.

The third generation of NS simulator started in July 2006. If NS3 is, as its predecessor, based on C++, OTcl is neglected in favor of C++ (network models) and the Python language (optional). In addition, it incorporates GTNetS [11], a simulator that is known for its support of scalability. These choices were made at the expense of backward compatibility that involves the manual and complete rewriting of any model developed under NS2. This incompatibility explains the sustained use of NS2 for which many protocols exist. [12] details more differences between these two generations.

Second well-known simulator in this category (while technically it is an all-around simulation environment based on discrete events), OMNeT++ [8] is a simulator adopting a modular approach developed in a graphical Integrated Development Environment (IDE) based on Eclipse for development, creation, configuration, execution and analysis results. OMNeT++ is composed of modules that communicate through messages. OMNeT++ provides the infrastructure to assemble the simulations of models and manage their configuration through a specific language named NED (NEtwork Description). OMNeT++ was designed to overcome the development problems in NS2 [13] [14] and is becoming even more popular. Often compared, they are the two most widely used simulators in the world of WSN [14]. Many WSN simulators are based on OMNeT++, like Mixim [15] (formerly Mobility Framework) -dedicated to the simulation of wireless network and mobile- or Pawis [16].

The problem is these interesting network simulators are not sensor platform-oriented and they are thus too high-level for hardware considerations. Moreover, there is no separation between computation and communication models. That modeling is not suitable for hardware analysis and explorations. Then, such simulators do not have accurate energy models [17], whereas it is the main constraint in WSN.

Node simulators refer to precise hardware descriptions, with a synchronization strategy among the nodes, such as Avrora [18], TOSSIM [19], powerTOSSIM [20], Sycyphos [21] or SCNSL [22]. These simulators are well suited for embedded system designs analysis, requiring precise low-level models.

TABLE I. SIMULATION PLATFORMS AND HARDWARE HETEROGENITY SUPPORT

Simulator	Language	Hardware modeling	Heterogeneity support
NS2	C++, OTcl	No	Yes
OMNeT ++	C++	No	Yes
Avrora	Java	Yes (limited to ATMEL)	No
TOSSIM	C	Power TOSSIM: limited to ATMEL	Yes
Sycyphos	SystemC	Yes	No
SCNSL	SystemC	No	No

Avrora [18] is a sensor network instruction set simulator (written in Java). It combines the precision of ATEMU [23] (cycle accurate) to the scalability of TOSSIM (up to 10,000 nodes). Avrora is furthermore language independent and of the embedded operating systems. The disadvantage of such a tool is its hardware support limited to ATMEGA128 architecture from ATMEL (node MICA and MICAZ). Moreover, using a high-level language, Avrora cannot be easily integrated into a conventional hardware design flow.

TOSSIM [19] and PowerTOSSIM [20] can emulate the execution of TinyOS. The application code of TinyOS is compiled and taken into account in the simulation framework. TOSSIM can consider thousands of TinyOS nodes with a very fine granularity. PowerTOSSIM is an extension of TOSSIM that gives power consumption evaluation. The main problem of these frameworks is that the user is constrained to a specific platform (typically MICA motes) and a single programming language (typically TinyOS/NesC) [24].

Sycyphos [21] objective is to enable design at system level down to circuit-level, with the help from multilevel simulation. Sycyphos is dedicated to power consumption evaluation and reliability study. It is based on Transaction Level Modeling (TLM), and uses multi-master bus architecture for radiofrequency network modeling. Nodes models are based on a multi-threaded instruction set simulator.

SCNSL (SystemC Network Simulation Library) [22] is an event-driven simulator of networked embedded systems, written in SystemC and C++. As SystemC is a C++ class library, it has the advantage to model both hardware and software. SystemC is a classical and widely used modeling language in micro-electronic systems design and particularly in System-On-Chip design.

Table I gives an overview of the most known simulators, it details their modeling language, if hardware is modeled, and if simulators support heterogeneous nodes (different hardware) simulation. The analysis of Table I leads to the conclusion that there is no simulation platform taking hardware into account (electronics designer level) and at the same time supporting heterogeneous (hardware different) nodes in the same network. Based on this conclusion, we planned to answer this problem.

Even with no support on hardware details and heterogeneity, SCNSL demonstrates a great perspective for accurate system-level simulation of WSN systems, and its architecture and language are well suited. Indeed, SCNSL models include nodes and network separately. That permits a low level modeling, with hardware support, and an easily scalable and tunable architecture. By our opinion, it also could answer fine granularity modeling, fine and accurate power consumption analysis and heterogeneous support.

Meanwhile, limitations of that library are numerous. We detail some of them. The "node" block models at once the hardware node (microcontroller and radiofrequency unit); therefore, its behavior does not reflect real hardware. Moreover, only a subset of the IEEE 802.15.4 standard is implemented in this alpha version: unslotted CSMA-CA policy with acknowledgments. Then, simulation result is a CPU time; important node-level and network-level results are not calculated. SCNSL includes three modules: node (SystemC), node_proxy (SystemC) and network (C++), as shown in Fig. 2.

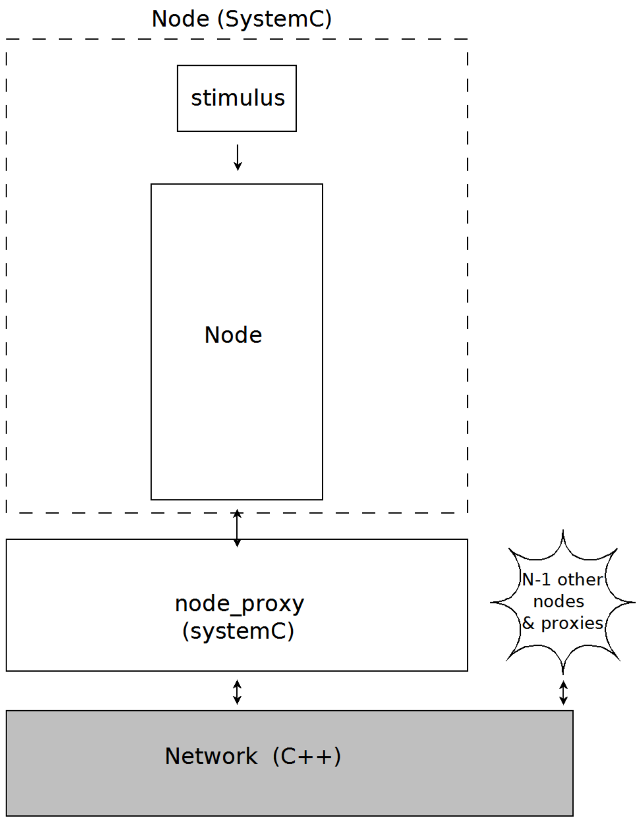


Figure 2. SCNSL model architecture

During the initialization of the simulation, each node registers its information (e.g., location, TX power and RX sensitivity) to a network class, which maintains the network topology and transmits packets to other nodes. The node_proxy is an interface between the network and nodes. By using node_proxy, nodes can be designed as pure SystemC modules so as to exploit all advantages of SystemC in hardware/software co-design and verification.

Our simulation platform is based on SystemC and C++, and SCNSL architecture was the starting point of our work.

III. IDEA1 SIMULATOR

A. Model architecture

The architecture of our model is close to real node hardware architecture, as Fig. 3 (compared to Fig. 2) shows. It includes sensor, microcontroller and radiofrequency unit blocks. Hardware, software and the whole IEEE 802.15.4 standard with many configurations are modeled. The SystemC blocks connected through a C++ network model was kept. The network model was modified to consider free space propagation. This simple propagation model could be extended to indoor context for example. Complex components, such as microcontroller or radiofrequency unit, are modeled as a Finite State Machine (FSM). Computing a Finite State Machine model in TLM with the efficient event-driven kernel simulator of SystemC is an interesting approach to reach fast simulation. It is the reason why IDEA1 is faster when compared to others simulators, like NS2.

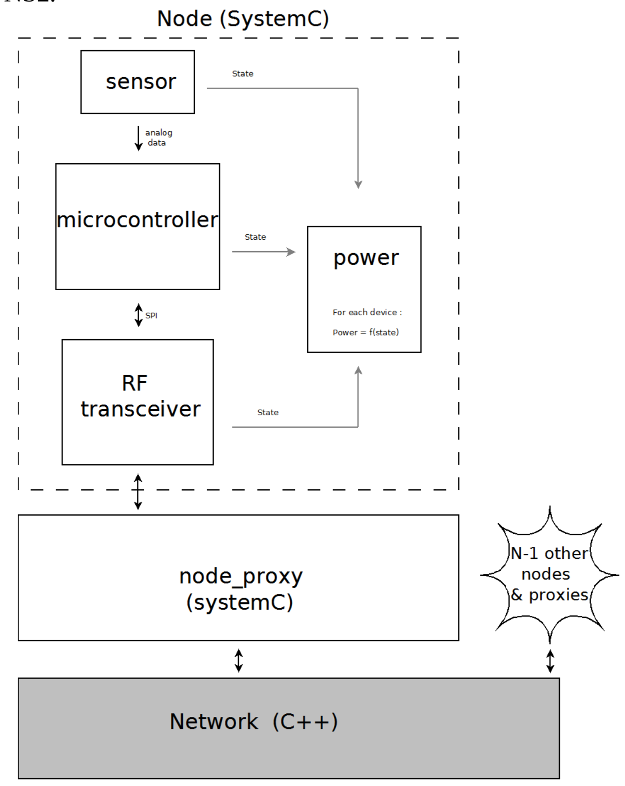


Figure 3. IDEA1 model architecture

B. Hardware and Software models

The sensor block receives physical data from a file, and sends its output voltage to the microcontroller. The sensor, microcontroller and radiofrequency unit are modeled separately, so that designers can easily switch these inter-changeable devices. These two parts communicate through SPI (Serial Peripheral Interface) interfaces.

The microcontroller is the central unit for processing and controlling purposes. In our typical case, the microcontroller initializes the radiofrequency transceiver, it reads (converts) analog data from the sensor, and communicates (digital) data with radiofrequency transceiver. As SystemC is event-driven, it is possible to configure events in the sensor, and make the node react to the sensor with hardware interrupts available in the microcontroller.

Switching between architectures is done by changing some parameters in the configuration files. The microcontroller model can for example switch from ATMEL to Microchip or Texas Instruments' ones. Radiofrequency unit can be Microchip or Texas Instruments devices. Figs. 4 and 5 show Finite State Machine examples for microcontroller. Parameters depend on the microcontroller itself and on the radiofrequency unit (for example if hardware support of IEEE 802.15.4 is present or not).

In the first case (Fig. 4), the microcontroller has to perform few tasks, as the radiofrequency unit is a relatively autonomous circuit: once configured, it is able to manage packet sending, packet reception or acknowledgments alone. The microcontroller has therefore to read the analog to digital converter, and send the data to the radio frequency circuit. In the second case (Fig. 5), the microcontroller is connected to a simple radiofrequency unit that just modulates ready-to-send data.

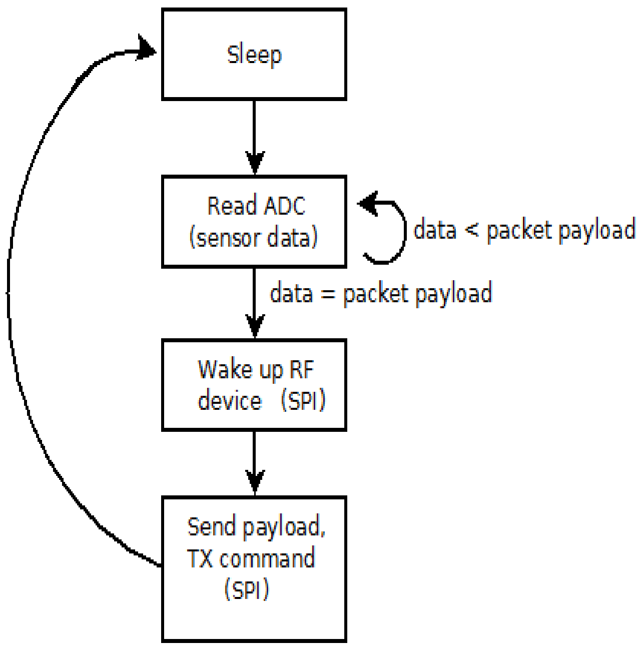


Figure 4. FSM of a microcontroller connected to a smart RF unit

The microcontroller must ensure all tasks, such as the composition of the packet (encapsulation of the data), or the waiting time for access to the channel (CSMA-CA mechanism) that depends on the channel load. Choice of the devices thus largely affects timing, communications and power consumptions.

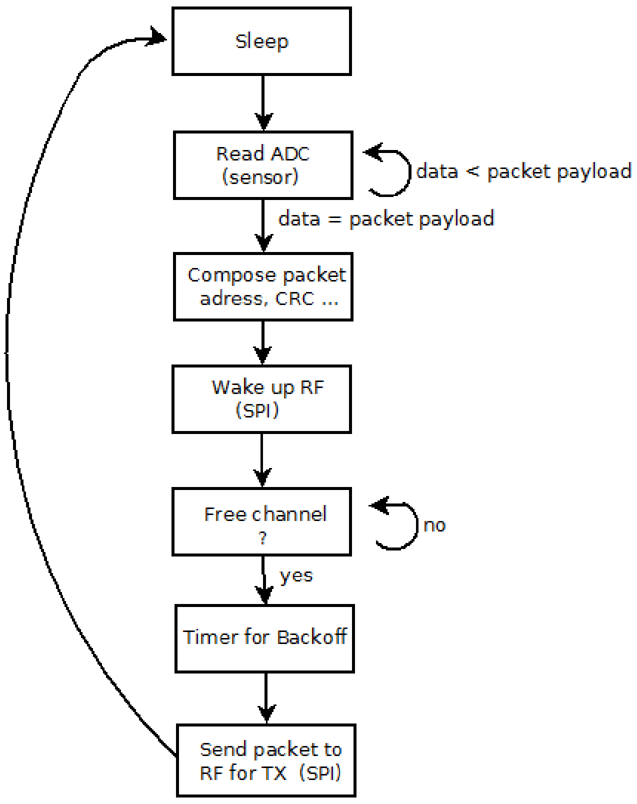


Figure 5. FSM of a microcontroller connected to a basic RF unit

In Finite State Machine, states are annotated by their duration and their power consumption. These values come from devices datasheets, and are all validated by measurements in our model implementation methodology. In order to have more accuracy, the CPU activity is considered. Fig. 6 shows the classical model that reflects the hardware part: sensor, microcontroller and radiofrequency units. The power module receives the current state of devices, and records all the state changes and timing in order to calculate and to log the power consumption. Energy can thus be evaluated with this power module. Table II details part of the lookup table that is implemented in power module (for ATMEL ATMega 128 and Texas Instruments CC2420 devices). All the devices in the library are modeled in this way.

The sensor and radiofrequency units are passive (basic) parts or active hard-coded, and their timing are well known. Meanwhile, the microcontroller has a more detailed finite state machine because of the (user) software that is running.

TABLE II. POWER INFORMATION OF ATMEGA128 AND TI CC2420

ATMega128 microcontroller		**CC2420 RF transceiver**	
Mode	Consumption	Mode	Consumption
Active	27 mW	Sleep	60 µW
Power Save	26.7 µW	Idle	1.28 mW
Power Down	0.9 µW	RX	56.4 mW
		TX (0 dBm)	52.2 mW
		TX (-1 dBm)	49.5 mW
		TX (-3 dBm)	45.6 mW
		TX (-5 dBm)	41.7 mW
		TX (-7 dBm)	37.5 mW
		TX (-10 dBm)	33.6 mW
		TX (-15 dBm)	29.7 mW
		TX (-25 dBm)	25.5 mW

Indeed, this software -often written in assembly or C language- can change, and thus behavior and timing of microcontroller. This software is analyzed with an Instruction Set Simulator (ISS) we have developed for a better integration in our platform. Our ISS calculates durations of all the functions. Whatever the function that is called, even by a hardware interrupt, it is taken into account in terms of timing and power consumption. Processing states in the finite state machine are thus accurate. This ISS was developed for several hardware architectures: ATMEL AVR ATMega and Texas Instruments MSP430 for the moment.

Owing to the fact that ISS are time-consuming simulators, we did not choose a co-simulation method; hence, the ISS does not run in parallel with the SystemC kernel. Indeed, the ISS runs once at the beginning of the simulation, and code is analyzed in order to calculate tasks timing. These timings are then associated with the finite state machine, as Fig. 6 shows.

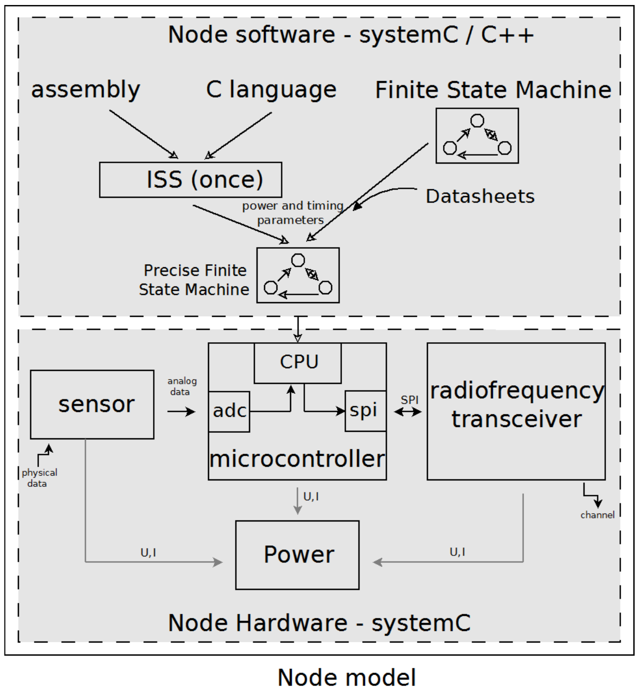


Figure 6. Node model including software for more accuracy

In detail, the ISS we have coded is an instruction set simulator that targets multiple hardware architectures. The whole instruction set of each targeted microcontroller is taken into account. It is written in C++ to offer compatibility support with our SustemC / C++ simulator. The ISS takes as input the ELF file produced by a compiler, often a C compiler. Next, it decodes the ELF file, looks which instruction is currently in scope and starts executing the functionality. At the end, the ISS produces an output file consisting of a lookup table pair: function name - number of corresponding clock cycles. ISS is also ran only once before SystemC simulation. More details on this ISS can be read in [25]. It is the main difference with classical ISS, that classically run in parallel with the main simulation kernel. Classical ISS thus slow down drastically the simulation speed. Using this lookup table and knowing the clock frequency of the microcontroller, these cycles are translated into timings. Once inserted in the SystemC simulation, software states in the finite state machine are timed, so a precise finite state machine is set.

Radiofrequency units are modeled individually because of their complexity and wide differences (that would make difficult a generic FSM). In Fig. 7 and Fig. 8 below, two FSM examples are drawn, of two well-known IEEE 802.15.4 compliant radiofrequency units: T.I CC2420 and Microchip MRF24J40.

As a whole, several sensors, microcontrollers and several radiofrequency units can be selected; the current library is detailed in Table III. Each sensor, microcontroller and radiofrequency unit can be mapped to each other. Each compliant radiofrequency transceiver includes the whole IEEE 802.15.4 standard.

Due to its architecture and file organization, the models library is easy to extend: new files, containing new models, are added in the folders, the main file includes them. C language #define statements permit to change the modeled hardware. Signals between modules are connected in the SystemC model, as it would be in real hardware.

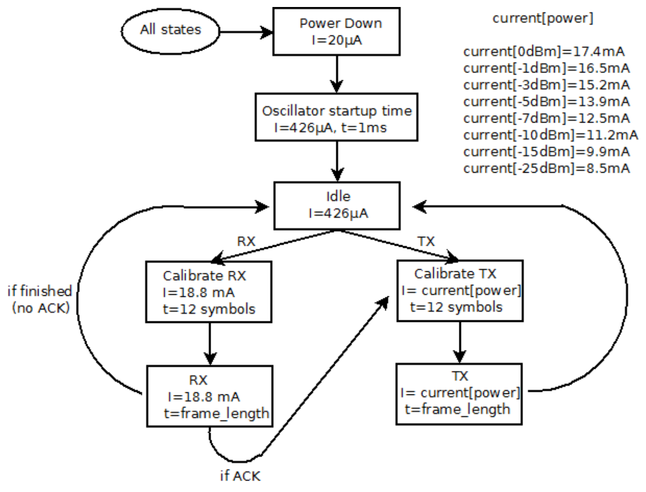


Figure 7. TI CC2420 simplified Finite State Machine

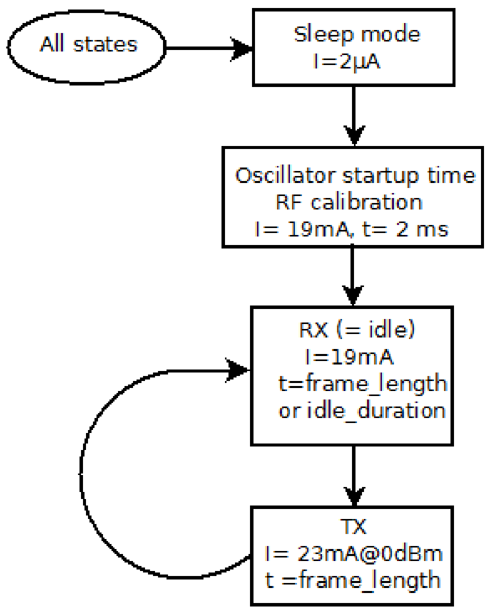


Figure 8. MRF24J40 simplified Finite State Machine

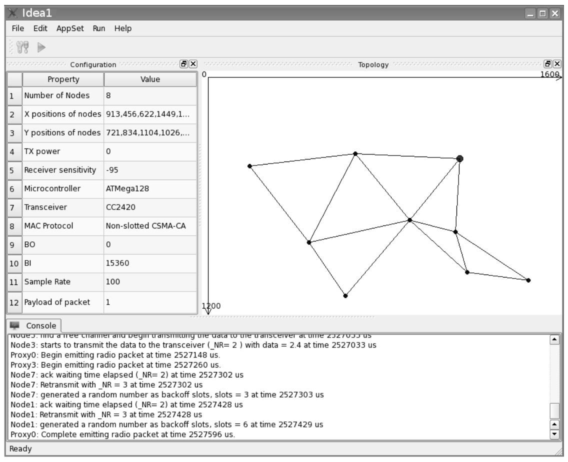
As it was previously published, all of these models were validated with experimental measurements on many test-beds [26], as detailed in Section IV.

TABLE III. MODELED HARDWARE DEVICES IN SIMULATOR LIBRARY

Sensor units	Microcontroller units	Radiofrequency units
N.S. LM35DZ Clairex CL9P4L	ATMEL ATMega128 Microchip 16LF88 T.I. MSP 430	T.I. CC2420 T.I. CC1000 Microchip MRJ24J40

C. *The simulator user interface*

The presented models can be used to simulate wireless sensor network communications at system level. To help SystemC / C++ non-specialists to use easily the simulation tool, we developed a graphical interface that is shown in Fig. 9.



	Property	Value
1	Number of Nodes	8
2	X positions of nodes	913,456,622,1449,1...
3	Y positions of nodes	721,834,1104,1026,...
4	TX power	0
5	Receiver sensitivity	-95
6	Microcontroller	ATMega128
7	Transceiver	CC2420
8	MAC Protocol	Non-slotted CSMA-CA
9	BO	0
10	BI	15360
11	Sample Rate	100
12	Payload of packet	1

```
Node3: find a free channel and begin transmitting the data to the transceiver at time 2527033 us
Node3: starts to transmit the data to the transceiver (_NR= 2 ) with data = 2.4 at time 2527033 us
Proxy0: Begin emitting radio packet at time 2527148 us.
Proxy3: Begin emitting radio packet at time 2527260 us.
Node7: ack waiting time elapsed (_NR= 2) at time 2527302 us
Node7: Retransmit with _NR = 3 at time 2527302 us
Node7: generated a random number as backoff slots, slots = 3 at time 2527303 us
Node1: ack waiting time elapsed (_NR= 2) at time 2527428 us
Node1: Retransmit with _NR = 3 at time 2527428 us
Node1: generated a random number as backoff slots, slots = 6 at time 2527429 us
Proxy0: Complete emitting radio packet at time 2527596 us.
```

Figure 9. Simulator graphical user interface

The user interface is composed of different sub-windows. A graphical viewer shows spatial position of nodes and the lines between nodes represent the possible communications according to locations, power of the transmission and sensitivity of the receiver. Hardware parameters are some of selectable microcontrollers and radiofrequency units. One of the many IEEE 802.15.4 configurations (in slotted or unslotted modes) and superframe parameters (SO, BO, BI etc.) can be selected. Sampling rate and payload of packets can thus be configured. User enters all parameters though a configuration window, called from menus. A click on the launch button in the graphical interface launches a SystemC simulation in background. Simulation log is displayed in the bottom window of the graphical interface, and a timing trace (Value Change Dump format: VCD) is created and can be opened. Output log files are thus generated for deeper analysis.

IV. Classical results and related work

From these log files, we can explore design space for the best solution (often the lower latency, best packet delivery rate, and the lower energy consumption). Many output curves are accessible: packet delivery rate (PDR), packet latency, node power consumption and energy per packet. All these results were validated with measurements on a 9 nodes network [27] with a TDMA-based GTS algorithm. These nodes, called N@L, are composed of Microchip devices: PIC16LF88 microcontroller and MRF24J40 radiofrequency unit. Each of the 8 nodes senses periodically a data and tries to send it to the coordinator. This period (sample rate) is the parameter for this study. Non-periodical scenario can be configured as well, timing is simply defined sequentially in the testbench file.

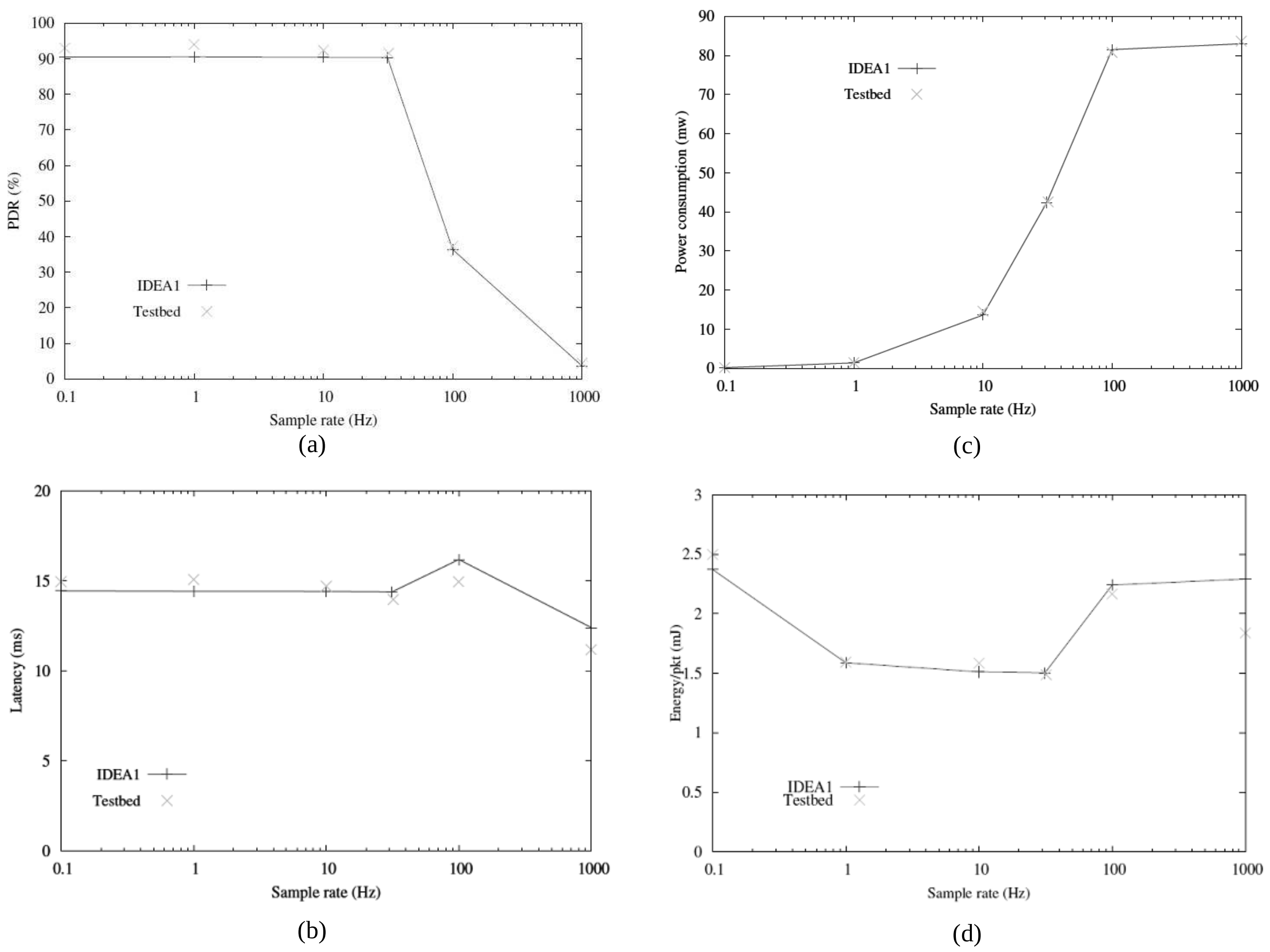


Figure 10. IDEA1 simulation and testbed measurements. Typical output curves: packet delivery rate PDR (a), packet latency (b), node power consumption (c), energy per packet (d).

IDEA1 simulation results are within 6% of the actual value obtained from real measurements. This good accuracy is not surprising since models are based on devices datasheets. Simulations and measurement simply validate datasheets.

Moreover, these results were compared to NS2 that we considered as a reference for this study. As our results are measurement-validated, we could explore accuracy of NS2 as well. NS2 is accurate for network-level results, such as packet delivery rate or latency. Indeed, hardware components have a small impact on these delays according to framing spacing and packet length compared to electronics components delays (software were taken into account at the same level in both simulators for this comparison). Meanwhile, the simulators have different results for energy per packet consumption, as Fig. 11 shows. This difference is especially important for low data-rate applications. Power consumption between IDEA1 and NS2 ranges from 9% to 16% in a non-beacon CSMA-CA algorithm. A simulation time analysis is shown in Fig. 12 where scalability is detailed. Fig. 12 presents relative simulation time: simulation time over simulated time. Even if both simulators are event-driven, Fig. 12 shows that IDEA1 kernel with FSM-based modeling takes a better advantage than NS2 on the application discrete behavior: IDEA1 curve is much more constant than NS2' one. Scalability is also better. Indeed, in low data rate scenario (typical WSN case), few events appear; simulator also simulates idle or sleep states. IDEA1 is 3.3 times faster than NS2. NS2 is more interesting in high data rate scenario (typical networked-computers case) because the ratio decreases. Anyway, ratio of IDEA1 decreases too, and it is still 3.1 times faster at 1000 Hz sampling rate.

Moreover, we showed that IDEA1 is able to provide a fine and precise power consumption analysis over many solutions: [27] detailed –for all IEEE 802.15.4 configurations- active and sleep consumptions of radiofrequency unit and microcontroller.

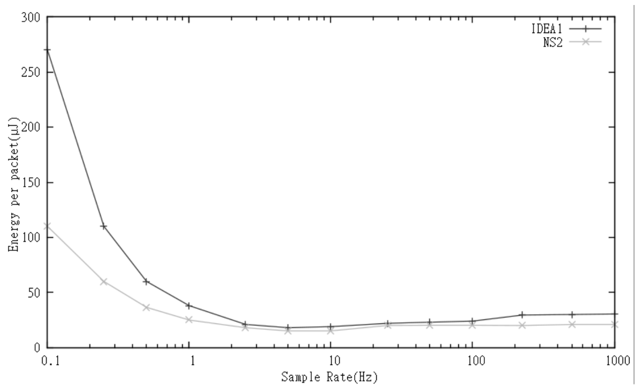


Figure 11. Node energy per packet. IDEA1 and NS2 simulations in non-beacon CSMA-CA.

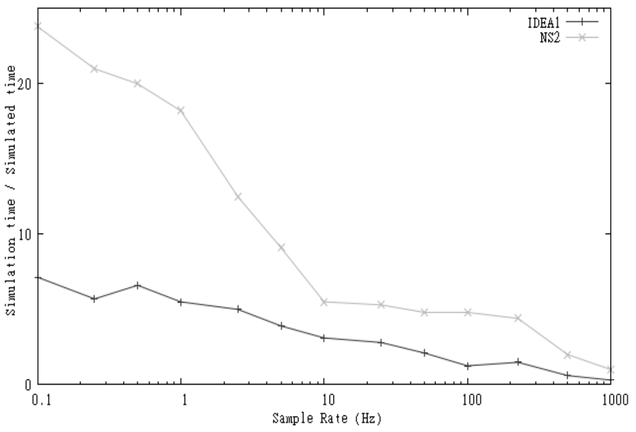


Figure 12. Relative simulation time (simulation time / simulated time). IDEA1 and NS2 simulations.

Fig. 13 shows this result. For two separate nodes, energy of radiofrequency unit in active mode (EnergyTransActive) and sleep mode (EnergyTransSleep) is detailed. In microcontroller, energy of internal hardware blocks (CPU, EnergyCPUPerNode, SPI communication block EnergySPIPerNode, analog to digital SAR converter EnergyADCPerNode) are monitored.

All these above results were obtained for homogeneous networks, so a single node hardware architecture.

The section below presents new simulation results in a heterogeneous network context.

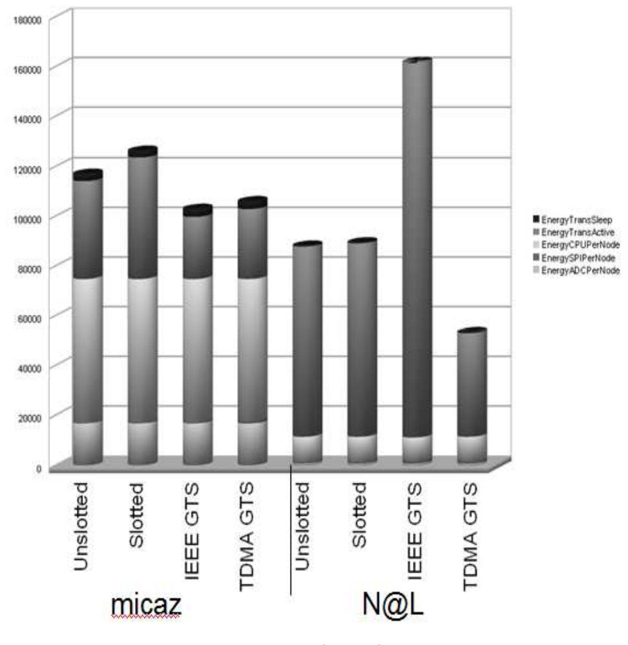


Figure 13. Energy consumption of radiofrequency transceiver and microcontroller internal blocks for two different platforms (μJ)

V. HETEROGENEOUS SIMULATION RESULTS

Heterogeneous support in simulators with fine and accurate hardware and software models is necessary, but few simulators support this feature, like [28]. One reason is the need of a complex instantiation of models.

Typical heterogeneous nodes are detailed in Fig. 14: node A and node B have different hardware devices. In our simulation, microcontrollers, and radiofrequency units are different (brand and model).

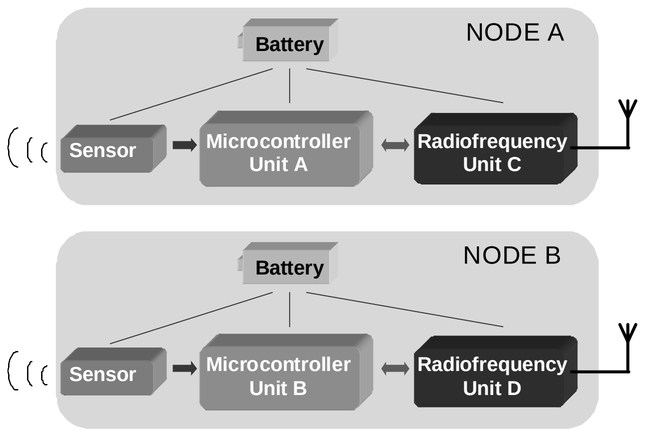


Figure 14. Typical node architectures in a Wireless Sensor Network (heterogeneous network)

As a test example, we simulated a 9 nodes network: one coordinator and eight nodes composed of Microchip PIC16LF88 and ATMEL ATMega128L microcontrollers and Microchip MRF24J40 and Texas Instruments CC2420 radiofrequency units, as specified in Table IV.

TABLE IV. NODES DEVICES FOR TESTBED AND SIMULATION

WSN device	Microcontroller unit	Radiofrequency unit
Coordinator	ATMega128	CC2420
Nodes 0..3	PIC16LF88	MRF24J40
Nodes 4..7	ATMega128	CC2420

Nodes sense the environment periodically every second, and transmit data over the network. Each transmission (packet) includes two data bytes (payload). Sensor nodes enter sleep mode as long as they can; the coordinator is always awake. The IEEE 802.15.4 non-beacon CSMA-CA communication scheme with no acknowledge is used, but all of the IEEE 802.15.4 can be configured for wider exploration. Simulation of this testbed gives a VCD trace, an extract is shown in Fig. 15. We can observe the coordinator's and nodes' microcontroller and radiofrequency unit states (R: Receive, T: Transmit, A: Active, S: Sleep CooMCUState stand for coordinator microcontroller state, Cooradiostate is the coordinator radiofrequency state. For classical nodes, states of microcontroller and radiofrequency unit are also detailed with mcustate0 and radiostate0 for node 0 and mcustate7 and radiostate7 for node 7. In this example, coordinator microcontroller is always active (A). At time 1065ms, coordinator radiofrequency unit sends a packet (T), node0 radiofrequency unit is in receive mode (R), node7 is in power down mode (0). Then, radiostate0 sends an acknowledgement (T), and then enters sleep mode. As no more processing is required, microcontroller of node 0 enters sleep mode. Node 7 wakes up at 1066ms. After a calibrating phase, microcontroller is active; radiofrequency unit is in receive mode. At 1066.5ms, microcontroller samples a data, sends it over SPI. After CCA, radiofrequency unit sends the data (T), and enters power down at 1069.3ms. Microcontroller enters sleep mode too, node 7 is totally in sleep mode too. ….). It is possible to monitor more signals in order to see for example the wireless channel usage, or the data transfer from the sensor to the radiofrequency unit through the microcontroller on each node, and data from the radiofrequency unit to the microcontroller on the coordinator.

Information in the log file gives a lot of output data, as packet delivery rate (PDR), and latency. Moreover, log file includes energy of each block of each circuit in each node. It is also possible to draw graphs such as the following ones. Fig. 16. presents the overall energy consumptions of the nodes.

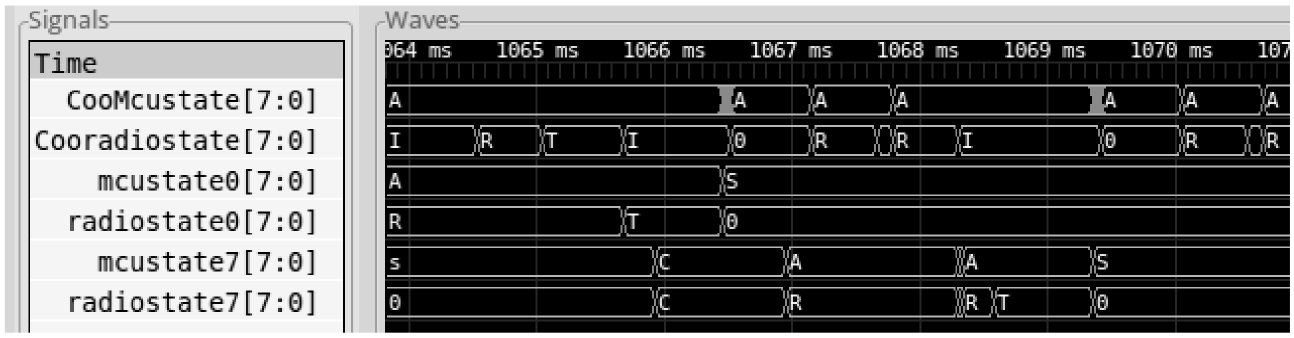


Figure 15. Extract of the output VCD file, focus on coordinator and nodes 0 and 5 (microcontrollers and radiofrequency units states)

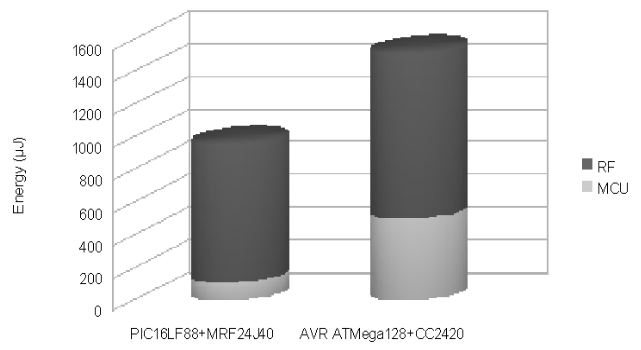


Figure 16. Heterogeneous nodes energy consumption

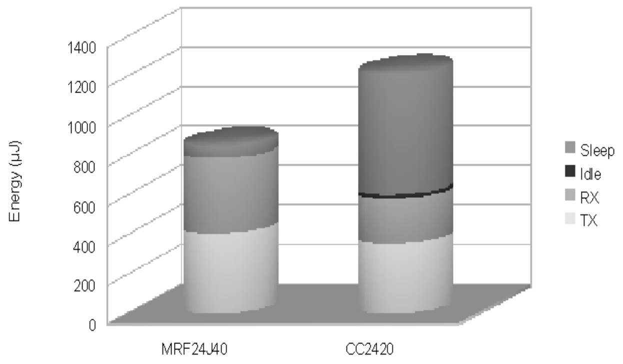


Figure 18. Radiofrequency units energy consumption comparison

Energy partitioning between the microcontroller and the radiofrequency unit for two heterogeneous nodes (node 0: Microchip PIC16LF88 and MRF24J40 and node 5: ATMEL AVR ATMega128 and T.I. CC2420) are shown. We can see the energy consumed by microcontroller (MCU energy in grey) compared to the radiofrequency unit one (RF energy in dark). In detail, PIC16LF88 consumes a total energy of 109µJ, AVR ATMega128 consumes 498µJ, so a 4.5 ratio. MRF24J40 consumes 848µJ, whereas CC2420 consumes 1016µJ, so a 1.2 ratio. This testbed shows an interesting combination of circuits that composes node 0, because it embeds the two most energy-aware circuits. Meanwhile, it is interesting to detail this big difference.

It is possible to have finer granularity and to detail the energy consumption of each block within hardware devices. Fig. 17 shows the microcontroller energy spent during (from top to bottom in bars) sleep, idle and SPI communications states. It is to note that CC24220 radiofrequency unit (with no IEEE 802.15.4 hardware support) has an impact on the active state duration of the microcontroller. Indeed, in that example, the CC2420 transceiver just modulates the packet; microcontroller implements the IEEE 802.15.4 standard by software. For example, it has to check for free channel, to respect delays (backoffs), to generate IEEE 802.15.4 compliant packets, to acknowledge if it is activated, etc. More SPI communications are thus required. This fact is visible on Fig. 17: active and SPI communication energy consumptions are important on AVR ATMega128.

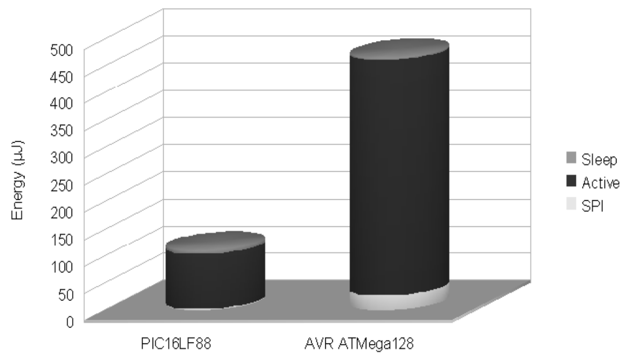


Figure 17. Microcontroller energy consumption comparison

On the other hand, the MRF24J40 transceiver is a more autonomous circuit, as it supports all the aforementioned aspects of IEEE 802.15.4 by hardware, the microcontroller is thus less active.

With the same fine granularity, it is possible to detail states of radiofrequency units, as shown in Fig. 18.

This figure shows it is possible to monitor energy consumed during states (from top to bottom on bars) of each radiofrequency unit: sleep, idle, receive (RX) and transmit (TX). Although sleep mode is the less power consuming, it is the longest state. Testbed is typical in WSN: duty cycle (wake-up duration / application period) is low. CC2420 has important energy consumption in sleep mode (compared to MRF24J40) because its power consumption is 8.5 times bigger. Sleep mode durations depend on activity of nodes, node 5 (AVR ATMEga128 + CC2420) needs more processing because of the basic radiofrequency unit, as discussed above. It is also meaningful to obtain a 10 ratio on energy consumption compared to node0. We can remark that MRF24J40 has no idle state; default state is RX (Fig. 8). While communicating or processing a packet, MRF24J40 is in RX state, it is why RX state is so energy consuming. As CC2420 has a lower power consumption in TX mode (52.2mW at 0dBm) compared to MRF24J40 (69mW at 0dBm), CC2420 has a lower energy consumption to transmit the same amount of packets.

We can see it is possible to optimize total energy with such a deep exploration.

VI. CONCLUSION

In this paper, heterogeneous support of IDEA1, our system-level simulator for Wireless Sensor Networks, was presented. This simulator is written in SystemC and C++. SystemC combines advantages of being a widely-used language in micro-electronic systems design flow, and permitting hardware and software co-modeling. Moreover, its kernel is efficient, and as our models are based on Finite State Machines, less events appear and simulation speed is fast compared to other simulators. The simulator graphical user interface permits configure easily a network and set the sensor nodes characteristics Simulation gives easy-to-read waveforms and easy-to-process output logs. IDEA1 library contains many hardware devices and the whole IEEE 802.15.4 standard. We demonstrated that it is possible to run quick and accurate simulations with different hardware devices on the nodes. Classical network simulators outputs (packet delivery rate (PDR), packet latency) are supported; as well as accurate timing, and detailed energy consumption of hardware devices that are measurement validated. It is also possible to simulate and compare many scenarios and configurations in order to run design-space exploration for the best-suited and lower power solution. Current release of IDEA1 is publicly available at http://www.idea1.fr.

REFERENCES

[1] D. Navarro, M. Galos, F. Mieyeville, and W. Du, "Heterogeneous Wireless Sensor Network Simulation," Proc. Sixth International Conference on Sensor Technologies and Applications, SENSORCOMM, pp. 292-295, Rome, Italy, August 2012.

[2] M. Horton and J. Suh, "A vision for wireless sensor networks," Proc. IEEE Microwave Symposium Digest, 2005.

[3] C. Fortuna, "Why is sensor data hard to get ?," Proc. COIN-ACTIVE Summer School on Advanced Technologies for Knowledge Intensive Networked Organizations in Aachen, 2010.

[4] Crossbow technologies inc, Document Part Number: 6020-0049-01-Rev-A, "Stargate X-Scale processor platform;" http://www.eol.ucar.edu/isf/facilities/isa/internal/CrossBow/DataSheets/stargate.pdf. Last accessed: May 16th, 2014.

[5] Texas Instruments, "A Powerful System-On-Chip for 2.4-GHz IEEE 802.15.4, 6LoWPAN and ZigBee Applications," SWRS096A –December 2012 – Revised April 2013, http://www.ti.com/cc2538-pr-ds1. Last accessed: May 16th, 2014.

[6] W. Du, D. Navarro, and F. Gaffiot, "Towards a Taxonomy of Simulation Tools for Wireless Sensor Network," Proc. International Conference on Simulation Tools and Techniques, 2010.

[7] S. McCanne and S. Floyd, "Network Simulator NS-2," http://www.isi.edu/nsnam/ns, 2010. Last accessed: May 16th, 2014.

[8] A. Varga, "The OMNeT++ discrete event simulation system," Proc. European Simulation Multiconference, 2001.

[9] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," Computer Network journal, vol 52, pp. 2292-2330, August 2008.

[10] V. Naoumov and T. Gross, "Simulation of large ad hoc networks," Proc. 6th ACM international workshop on modeling analysis and simulation of wireless and mobile systems, New York, USA, 2003.

[11] G. F. Riley, "Large-scale network simulations with GTNetS," Proc. Winter simulation conference, pp. 676-684, 2003.

[12] J. L. Font, P. Inigo, M. Domínguez, J. L. Sevillano, and C. Amaya, "Analysis of source code metrics from ns-2 and ns-3 network simulators," Simulation Modelling Practice and Theory, Elsevier, Vol 19, issue 5, pp. 1330-1346, 2011.

[13] E. Weingartner, H. vom Lehn, and K. Wehrle, "A performance comparison of recent network simulators," Proc. IEEE international conference on communications, Dresden, Germany, 2009.

[14] C. Mallanda et al., "Simulating wireless sensor networks with OMNeT++," http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.331.6889&rep=rep1&type=pdf. Last accessed: May 16th, 2014.

[15] A. Kopke et al., "Simulating Wireless and Mobile Networks in OMNeT++, The MiXiM vision," Proc 1st international conference on simulation tools and techniques for communications, networks and systems & workshops, Simutools '08, Brussels, Belgium, 2008.

[16] J. Glaser, D. Weber, S. A. Madani, and S. Mahlknecht, "Power aware simulation framework for wireless sensor networks and nodes," EURASIP Journal on Embedded Systems 2008.

[17] F. Chen, I. Dietrich, R. German, and F. Dressler, "An Energy Model for Simulation Studies of Wireless Sensor Networks using OMNeT++," PIK - Praxis der Informationsverarbeitung und Kommunikation, vol. 32, issue 2, pp. 133–138, 2009.

[18] B. Titzer, D. Lee, and J. Palsberg, "Avrora: Scalable sensor network simulation with precise timing," Proc. Symposium on Information Processing in Sensor Networks, pp. 477-482, USA, 2005.

[19] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: accurate and scalable simulation of entire tinyos applications," Proc. of the 1st int. conf. on Embedded networked sensor systems, ser. SenSys '03, pp. 126-137, New York, USA, ACM, 2003.

[20] V. Shnayder, M. Hempstead, B. Chen, G. W. Allen, and M. Welsh, "Simulating the power consumption of large-scale sensor network applications," Proc. of the 2nd int. conf. on Embedded networked sensor systems, SenSys '04, pp. 188-200, New York, USA, ACM, 2004.

[21] J. Wenninger, J. Moreno, J. Haase, and C. Grimm, "Designing lowpower wireless sensor networks," Proc. Forum on Specification & Design Languages, Oldenburg, Germany, September 2011.

[22] F. Fummi, D. Quaglia, and F. Stefanni, "A SystemC-based Framework for Modeling and Simulation of Networked Embedded Systems," Proc. Forum on Specification and Design Languages, 2008.

[23] J. Polley, D. Blazakis, J. McGee, D. Rusk, and J. S. Baras, "ATEMU: a fine-grained sensor network simulator," First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, pp. 145-152, Oct. 2004.

[24] D. Weber, J. Glaser, and S. Mahlknecht, "Discrete event simulation framework for power aware wireless sensor networks," in Proc. of the 5th Int. Conf. on Industrial Informatics, pp. 335-340, 2007.

[25] M. Galos, D. Navarro, F. Mieyeville, and I. O Connor, "A Cycle-Accurate Transaction-Level Modelled Energy Simulation Approach for Heterogeneous Wireless Sensor Networks," 10th IEEE International NEWCAS Conference, Montréal, Canada, June 2012.

[26] F. Mieyeville, W. Du, I. Daikh, and D. Navarro, "Wireless Sensor Networks for active control noise reduction in automotive domain," Proc. 14th International Symposium on Wireless Personal Multimedia Communications, 2011.

[27] F. Mieyeville, D. Navarro, W. Du, and M. Galos, "Energy-centric simulation and design space exploration for Wireless Sensor Networks," Wireless Sensor Networks: Current Status and Future Trends, CRC Press, Taylor & Francis Group, November 2012.

[28] L. Girod, T. Stathopoulos, N. Ramanathan, J. Elson, D. Estrin, E. Osterweil, and T. Schoellhammer, "A system for simulation, emulation, and deployment of heterogeneous sensor networks," Proc. Int. Conf. on Embedded Networked Sensor Systems, 2004.