Worldsens: development and prototyping tools for application specific wireless sensors networks

Antoine Fraboulet, Guillaume Chelius, Eric Fleury

Worldsens: Development and Prototyping Tools for Application Specific Wireless Sensors Networks

Antoine Fraboulet
CITI / INSA de Lyon
INRIA / Compsys
F-69621 France

Guillaume Chelius
INRIA/ARES
CITI/INSA de Lyon
F-69621 France

Éric Fleury
CITI/INSA de Lyon
INRIA/ARES
F-69621 France

ABSTRACT

In this paper we present Worldsens, an integrated environment for development and rapid prototyping of wireless sensor network applications. Our environment relies on software simulation to help the designer during the whole development process. The refinement is done starting from the high level design choices down to the target code implementation, debug and performance analysis. In the early stages of the design, high level parameters, like for example the node sleep and activity periods, can be tuned using WSNet, an event driven wireless network simulator. WSNet uses models for applications, protocols and radio medium communication with a parameterized accuracy. The second step of the sensor network application design takes place after the hardware implementation choices. This second step relies on the WSim cycle accurate hardware platform simulator. WSim is used to debug the application using the real target binary code. Precise performance evaluation, including real-time analysis at the interrupt level, are made possible at this low simulation level. WSim can be connected to WSNet, in place of the application and protocol models used during the high level simulation to achieve a full distributed application simulation. WSNet and WSNet+WSim allow a continuous refinement from high level estimations down to low level real-time validation. We illustrate the complete application design process using a real life demonstrator that implements a hello protocol for dynamic neighborhood discovery in a wireless sensor network environment.

Categories and Subject Descriptors

C.2.2 [**Computer Systems Organization**]: Computer-Communication Networks; D.2.5 [**Software Engineering**]: Testing and Debugging—*Debugging aids, Monitors*; D.2.6 [**Software Engineering**]: Programming Environments—*Integrated environments*

IPSN'07, April 25-27, 2007, Cambridge, Massachusetts, USA.

General Terms

Algorithms, Design, Performance, Experimentation

Keywords

Sensor networks, Development, Simulation, Performance

1. INTRODUCTION

Advances and miniaturization of micro-electro-mechanical systems (MEMS) and micro-electronic devices have pushed the development of a new field of application for wireless networks: Wireless Sensor Networks (WSN). Sensor networks are composed of small objects which monitor their surrounding to measure environmental informations such as the temperature, air/water pollution level or to detect movements, vibrations, etc. Sensor nodes communicate through a radio device in order to manage the network and to gather the produced data. Monitoring stations (also called sink stations) are generally integrated in the network. They are used both to collect the data and to manage the network.

Sensor networks are constrained regarding resources: computation, communication as well as energy. In this context and in order to deploy efficient sensor networks, a specific care must be invested in the design of the application, communication protocols and operating systems that are to be executed, as well as in the code implementation and the hardware platform that are to be deployed.

Despite the relative simplicity of its basic components compared for example to a PC - a sensor node is composed of an integrated sensor, a processor, a radio and an energy battery - sensor networking offers a great diversity. Diversity in the hardware: MicaZ, Telos, SkyMote with ARM7, AVR or TI micro-controllers. Diversity in the radio and physical layers: 868MHz, 2.4GHz, modulation, frequency hopping. Diversity in the operating systems: TinyOS, Contiki, FreeRTOS, JITS. Diversity in the constraints: real-time constraints, energy constraints, memory or processing constraints. Diversity in the applications: from the military to the civil use. Almost all kind of data traffic patterns, mobility patterns or communication patterns can be encountered. Facing this variety of problems, there is no "one-fits-all" solution but rather one "better" solution for each specific case.

As a result of this diversity, the design, implementation and deployment of a sensor network application is a really complex task. Choices have to be made at each level of the design process and these choices generally have an impact on the application performances. Moreover, debug and profiling of micro-controllers and more generally distributed

embedded systems is clearly a hard task. As a consequence, there is a real need for a development framework that would give developers the opportunity to test and simulate their solutions at the different layers; from the high level design to the implementation through the hardware choices. Moreover, this development framework should not induce too much of an extra load such as the writing of the application or the constraints in various descriptive languages. The process should be as intuitive as possible.

In this context, Worldsens offers an integrated platform for the design, development, prototyping and deployment of applications for wireless sensor networks. This integrated platform consists in two simulators that are used at the different levels of the application development and that offer very accurate results on the application performances. It basically performs distributed simulations of the hardware platforms and radio medium with an instruction and radio byte accuracy. Following this introduction, we briefly introduce the currently available simulation tools for sensor networks and their limitations in section 2. Then, we present the Worldsens development framework in section 3. The simulation tools WSim and WSNet are described respectively in section 4 and 5. We explain how to use them conjointly in section 6. Then we show how Worldsens can be helpful in the development of an application in section 7. We finally conclude and present future works in section 8.

2. STATE OF THE ART

The main past efforts in simulating wireless sensor networks have focused on the protocol level, such as protocols and MAC issues [1], using models of the software and hardware. This approach relies on classical network simulation frameworks such as the ones used in the context of wireless and ad hoc networks (*e.g.* NS-2, Opnet, *etc*). However, in order to better understand the complexity of designing real sensor network applications, it becomes mandatory not only to model but also to offer a precise time accurate simulation of the network as well as the sensor node internals. Precise simulation of the node internals is necessary in order to study energy consumption as well as time-dependent properties such as the ones that may appear when using asynchronous components linked to the microcontroller (*e.g.*, RF component). A more complete overview of simulators for WSN can be found in [5] and [6].

2.1 Network simulation

The first tools that were used to simulate sensor networks are classical network simulation frameworks such as NS-2, OMNET++, GTSNetS, JiST, J-Sim, GloMoSim, *etc.* With these tools, sensor nodes are generally described using a layered architecture. These layers are responsible for modeling an aspect of the sensor node behavior, software or hardware. *E.g.* there is generally a mobility model, an application model, a routing model, a mac layer model, *etc.* For many reasons, these simulators are limited in their use.

First, they have a high-grained precision regarding the sensor node internals. As a consequence, some of their results such as a node energy consumption must be considered carefully.

Second, it is not the target application and the operating system that are simulated but a model of them. These simulators do not provide any hint about the application code.

For these reasons, these simulators are useless in the context of application debugging and profiling and helpless for implementation choices. Their interest lies at a higher level, during the application design process, for example during performance evaluation and dimensioning of network protocols and mac layers.

2.2 Node simulation

We know few simulators that intend to reduce the use of modeling in order to offer better accuracy: Tossim [14], Atemu [15], Avrora [16] (dedicated to Crossbow AVR/Mica2 motes) and Mantis [3]. Tossim simulates a complete TinyOS sensor network. It provides a high degree of accuracy by using model of few low-level components. It runs the software source code. However, the Tossim approach is not able to capture the fine-grain timing and interrupt properties of the code.

Mantis prototyping environment is dedicated to the Mantis OS and, as Tossim, does not reflect the time dependencies. Such fine-grain details can be very important when dealing with applications that depend and interact with the hardware, which is notoriously the case in sensor network applications.

Atemu was the first to offer an instruction-level simulator. Avrora is also an instruction-level simulator written in Java. Like Atemu, Avrora simulates a network of motes, runs the unchanged microcontroller code and performs simulation of the devices and the radio communication. One important limitation in both simulators is that they do not implement some principal components like the clock manager. So, the microcontroller cannot change its own frequency which is very limiting as dynamic frequency and voltage scaling are mechanisms that can highly reduce the power consumption. Avrora is not able to model a clock drift on a given node or between the different components of a given sensor node. It is not precise enough to perform communication debugging with external components. Note that Avrora does not model the node mobility.

Another limitation of Avrora is that it uses a multi-threaded approach so that all nodes are running on a single machine. Such an approach is scalable up to a given limit. Our goal is to offer a better scalability by using distributed simulations over several machines. Worldsens tools are able to perform high-level simulations that can be refined down to a very accurate timing precision and which can be used for debugging and performance evaluation.

3. THE Worldsens FRAMEWORK

3.1 The simulation tools

The Worldsens development framework consists in two simulators that are used either independently or in conjunction during the application development steps:

- WSim is a platform simulator. It relies on cycle accurate full platform simulation using microprocessor instruction driven timings. The simulator is able to perform a full simulation of hardware events that occur in the platform and to give back to the developer a precise timing and performance analysis of the simulated software.

- WSNet is a modular event-driven wireless network simulator. It may be used either alone to evaluate, refine

and validate the application high level design: choices of protocols, traffic pattern, application dimensioning, protocol parameters tuning and physical layer choices; or in conjunction with WSim to simulate a whole sensor network with a high accuracy.

These simulators are described with more details in sections 4 and 5.

3.2 The framework

A first characteristic of Worldsens is that once the design choices have been made, the simulation platform only handles the real application native code in order to test and validate the application at the instruction level. We do not want to impose to developers the task of rewriting their application in a particular description language or to transform low level parts of their code in order to be compliant with our simulation tools. In addition to this, we have designed our Worldsens simulation tools with respect to the following goals:

- Worldsens captures the behavior of the application at a low level: native instruction, bit, interrupt and byte radio level. It provides an accurate time control. The timing results of WSim have been validated with real-world systems (`http://www.worldsens.net/`) down to the clock cycle level;

- Worldsens is independent of any programming language and any OS as it runs the native code generated for the target microcontroller. WSim can be used to port and optimize an OS on a given target (*e.g*, TinyOS and Contiki runs inside WSim for the MSP430 target);

- Worldsens runs several node applications on several machines, offering flexibility and scalability. The synchronization between WSim and WSNet is crucial for the evaluation of real-time applications. Time constrained protocols issues are resolved by implementing a save/backtrack mechanism together with *rendezvous* points (RPs). WSNet is responsible for setting RPs during the simulation;

- Worldsens enables the use of profiling tools on code (*e.g.* kcachegrind [13]) as many traces can be collected (instructions, clock frequencies, interrupts, energy);

- Worldsens can monitor low power dynamic frequency scaling used in sensor nodes ;

- Worldsens reflects the behavior of the network. At the same time, one may refine the degree of accuracy of the radio medium simulation;

- In order to model the behavior of a global application, Worldsens handles large sensor networks, with hundred to thousand nodes.

A second characteristic of Worldsens is that all steps of the application development, *i.e.* the design process, the dimensioning, the validation and finally the profiling, are done using the same simulation tools.

3.3 The Development process

Worldsens proposes a development framework that follows the conception flow depicted in Figure 1. Based on the application specifications and constraints, the first step consists in doing the high level design of the application. This high level is used to make choices on the protocols and the distributed architecture of the application like, for example, the choice of the MAC, routing and dissemination protocols but also the physical layer characteristics (radio modulation and frequency). In a first simulation step, these choices are validated at a high level using the WSNet network simulator. The protocols are also dimensioned during this step. Depending on the traffic and mobility patterns, the application and the number of nodes, several protocol parameters must generally be tuned to achieve the best performance. At this step, the simulation process does not require to offer a precise understanding of the nodes internals but should rather offer statistical estimations on the performance of the design choices. These estimations concern in particular protocol efficiency like data delivery rate or latency but also energy consumption. The simulation results are used to refine the high level design until achieving satisfying results.

The second step consists in making implementation choices. This concerns both the software specifications, *i.e.* the operating system, the programming language, the application internal design, and the hardware choices, *i.e.* micro-controller, R/F transceiver. Once these choices are made, the software implementation may start. At this step, and after cross-compilation for the target platform, the code is simulated using WSim, a hardware platform simulator, and WSNet. WSim handles the target binary code and simulates the hardware platform at the instruction level. It offers a gdb interface as well as measures and informations on the platform behavior and the code performance: number and scheduling of interruptions, micro-controller states. Using this tool, the developer can debug and validate its implementation as well as verify real-time properties of its application. Using feedback from WSim, it can also profile its application for better performance and lower energy consumption. The coupling of WSim and WSNet is a powerful tool which allows the precise simulation of a whole sensor network at the radio byte and instruction level, where each node is simulated by a WSim program and the radio medium is simulated by WSNet.

4. WSim: AN INSTRUCTION BASED SIMULATOR FOR SENSOR NODE

The WSim hardware model granularity is derived from the sensor node printed circuit board. The simulator is composed of hardware block descriptions that match the chip level description of the system. Each block description is available as a software library within the simulation framework. A sensor node platform can be built by selecting the components description and by writing a single file that describes the physical interconnection between these blocks. Deriving a new sensor platform within the WSim environment consists in writing the missing hardware block descriptions with some simple guideline rules and the interconnection file. The hardware simulator uses timed finite state machines to describe the behavior of hardware blocks with a well defined API to connect the block to the different communications interfaces (GPIO ports, USART, SPI, . . .).

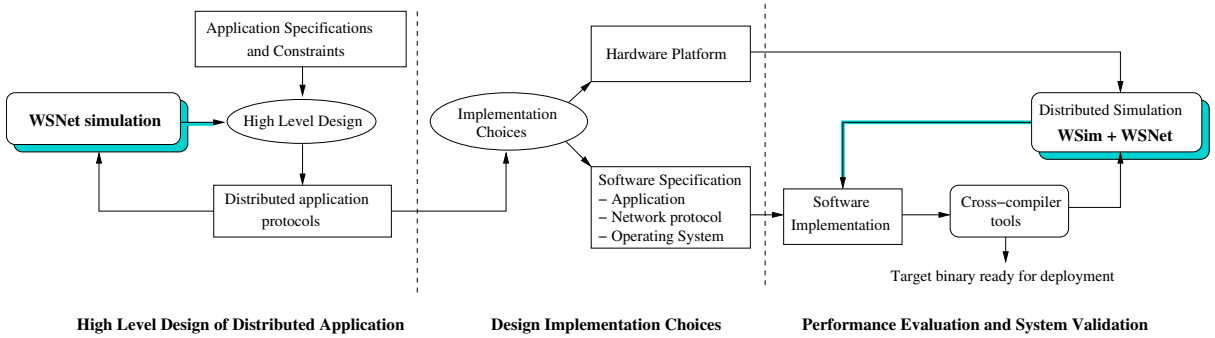

Figure 1: Worldsens Wireless Sensor Network Design Flow

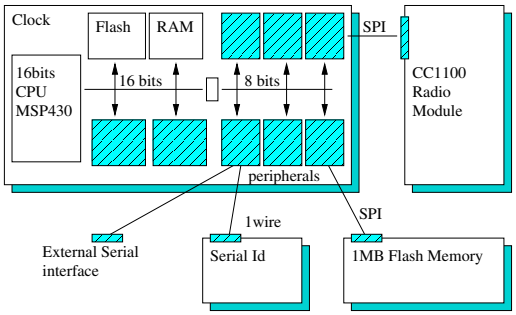

Figure 2: Example of a Wireless Sensor Node used in WSim

The simulation framework also includes a logger library to report selected system events and messages (*e.g.* missed peripherals interrupts that occur while interrupts are disabled in the microcontroller) or to catch some frequent programming errors that the block developper would like to report (*e.g.* sending a byte to an UART without proper initialization). WSim also supports a fast event tracer mechanism that saves to a file a selected signal activity. These traces can be used offline to study the runtime behavior of the sensor node with regards to timed events. Among the reported traces, the simulator can report events such as current clocks changes, interrupt reporting or serial port activity. This file can be used for offline performance analysis and timing validation among the platform peripherals. Finally, WSim has the ability to act as a GDB remote debugger that can be used for target code debugging using step-by-step mode with breakpoints and watchpoint as a conventionnal debugger.

Fig. 2 presents an example of node that can be fully simulated with WSim. This platform includes a full Texas Instrument MSP430f1611 with its complete instruction set and all digital blocks (timers, basic clock module, serial ports with UART and SPI modes, etc). The microcontroller is connected to external system peripherals: 1MB flash memory module (ST M25P80), serial id (Maxim DS2411), a Chipcon CC1100 packet radio interface, *etc.*

4.1 Cycle Accurate Simulation Timing

One of WSim main features is that the simulator can handle real target binary code. Application code instructions are used as a time reference to compute the local node simulation time according to the node configured frequency. This behavior conforms to the interrupt latency precision available on real hardware as interrupt can only be taken into account when the current instruction finishes. The time given by the instruction execution is then used as a reference for every other clock sources that are present in the platform.

This mechanism gives WSim the ability to simulate precisely several clock sources, each with its own frequency, simultaneously during the execution. This feature has several important properties such as:

- Multiple clock sources can be simulated in the system and each of these clocks can select an independant running frequency in different clock domains.
- Peripherals can use their own clock subsystem.
- The simulated microcontroller can use a variable frequency clock subsystem. This includes clock frequencies generated by internal digital clock oscillators such as the one available in the MSP430 microcontrollers and time to stabilize the oscillator when waking up from sleep modes.

Simulation time reference is computed using a nanosecond precision. This timing precision allows to have precise ratio between multiple clocks in order to achieve accurate synchronization. Simulation can also include an artificial drift between clocks. This drift can be parameterized to act as real hardware clock imperfections, either on the main clock system for real time-shift between two nodes or among different subsystems within the same node.

The current drawback of the current timing model is that there can be only one microcontroller, that act as a clock reference, present on a node model within WSim. This should not be a limitation to current state of the art wireless sensor node architectures as they are all using only one microcontroller or microprocessor.

4.2 Target Code Debugging

WSim includes several features to debug and analyze sensor nodes application code. The first one is related to debugging using the remote GDB protocol. WSim allows to attach a GDB process to the simulator that can single-step the simulation and have full access to the registers and memory. The step by step function includes all available platform clocks and peripherals. It is thus possible to debug interrupts and dynamic frequency scaling parts of the application in a cycle accurate environment. Figure 3 presents a screenshot of the Insight debugger connected to WSim.

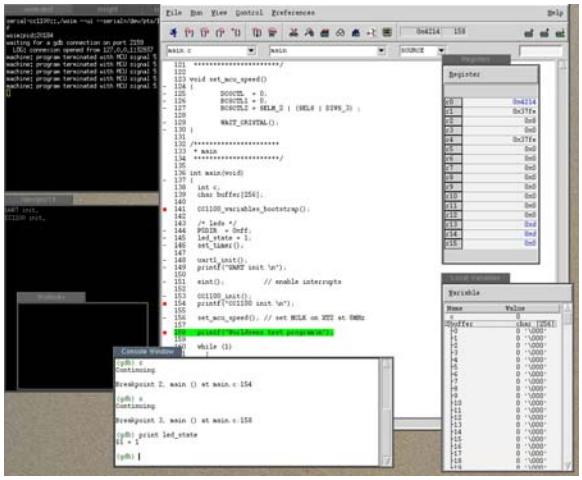

Figure 3: Insight debugger targeted for the MSP430 platform using the remote GDB protocol to control the simulation run

4.3 Node Timing and Performance Analysis

The tracer mechanism included in WSim allows to record events occurring during the simulation in order to give back to the programmer valuable offline performance estimations. Traces can be stored in several formats including Value Change Dump (VCD) that is used in several logic analyzers. Several types of events are reported such as interrupts arrival, lower power modes enter and exit time, controller frequency scaling and peripherals activity. Reporting all such events with their corresponding arrival time is a key feature for performance evaluation and software validation.

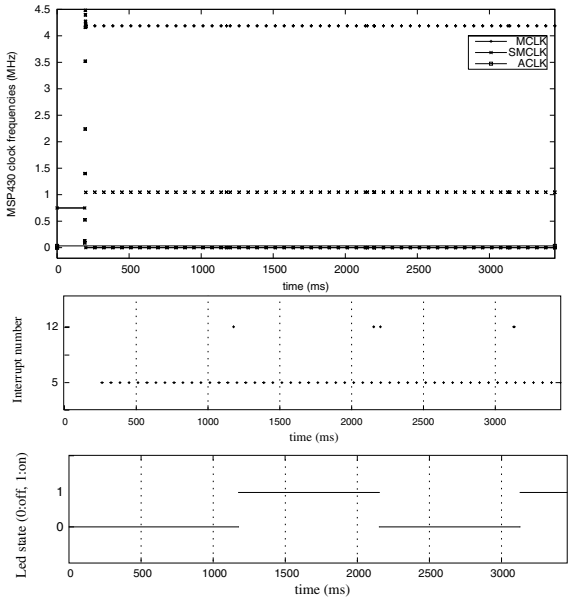

Figure 4: WSim traces for TinyOS Blink application

As an example, Fig. 4 has been obtained by running the TinyOS version 1.1 Blink example application on the node presented in Fig. 7. The Blink application starts up the operating system and uses a timer to change the state of a Led each second. By using WSim, we can record the full system activity. The presented results are the 3 main MSP430 clocks (ACLK: auxiliary clock, MCLK: master clock and SMCLK: sub-main clock), interrupt handlers entry and exit times and the red led state. The simulation presents the first 3.5 seconds of the running application.

Recording the exact behavior of the sensor node hardware allows to profile the target code very precisely. On the simulation example (Fig. 2), we can observe the TinyOS booting process in which the scheduler calibrates the internal oscillator to reach 4.1MHz for the MCLK clock. This clock is the master clock that schedules instruction execution in the microcontroller. The SMCLK is then switched to MCLK/4 to have a running frequency of 1MHz. This clock is used to drive peripherals such as serial ports. The ACLK clock is running at 32768Hz during the simulation. Interrupt 5 corresponds to the MSP430 timer A3 block while the interrupt 12 is used by the timer B7.

The first timer is used to schedule the operating system event loop `__nesc_atomic_sleep` function call while the second is driven by the application to be fired each second to switch on and off the led. Whenever the system detects that it can enter a sleep mode MCLK and SMCLK clocks are shut down and the processor is put into low power mode. ACLK clock is kept running to keep timers active. The middle figure of Fig. 7) shows that TinyOS still triggers an interrupt 5 every 62ms to check if some components have received new data on their input ports even if they are not used. This feature should be turned off in the present case to optimize energy. The led state is not changed each second but has a slight delay that can be also be seen on the interrupt 12 trace that comes from the timer interrupt multiplexer used within TinyOS. This timer multiplexer allows to have an arbitrary number of software interrupts running on top of the hardware. The interrupt engine of TinyOS uses a list of interrupts periods and recomputes, each time the hardware timer is fired, the next hardware timer delay that has to be scheduled to meet the next software interrupt. This computation is responsible for the introduction for the slight delay in interrupt scheduling.

By giving back to the designer all the necessary information, WSim allows to do a complete simulation analysis of the application code. When coupled with external peripheral traces, this analysis gives a full view of the *complete system behavior*. Deeply embedded real time application code can then be put to test before going to hardware platforms.

5. WSNet: A WIRELESS NETWORK SIMULATOR

WSNet is a modular event-driven wireless network simulator similar to the ones described in section 2.1. Its architecture consists in different blocks that model characteristics and properties of the radio medium and sensor nodes. It is described in Figure 5. As an example, separate blocks are defined for the radio propagation, the interference computation or the radio modulation. During one simulation, the behaviour of a block is specified using a model. A model is one particular implementation of the block functionnalities. As an example, if we consider the radio propagation block, one model implements the Friis free space [8] propagation model; another one implements a radio range model,

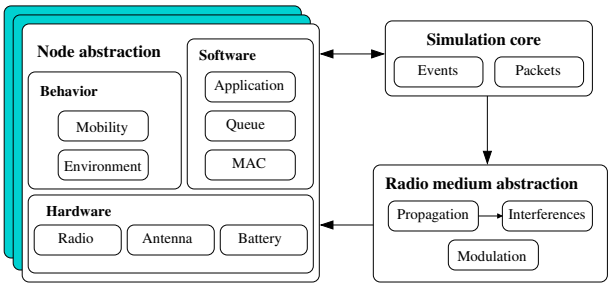

Figure 5: Block architecture of WSNet.

etc. Models are either provided with the simulator or developped by users.

In WSNet, nodes are either modeled using internal blocks or simulated by external programs that communicate with the radio simulator using a UDP/IP front-end. This design allows the setup of large distributed simulations using very accurate platform simulators. This would be later described in section 6

WSNet has been designed to offer a wide range of radio medium modeling, from a basic "perfect" physical layer with no collision, no path-loss and a fixed radio range to a very precise one: Rayleigh fading model, multiple radio resources (codes, frequencies), correlation factors between different radio frequencies and codes, additive interferences, complex radio modulations, complex antenna radiation patterns, *etc.*. Given this range of models, it becomes possible and very simple to test the behavior of a protocol in different radio environments. An interesting possible design process is to refine the dimensioning and optimization of a protocol by progressively increasing the radio modeling precision and complexity.

5.1 Radio medium modeling

WSNet models the radio environment using the three following blocks:

- a **propagation** model: *e.g. free space*, *rayleigh*, *etc.*;
- a **modulation** model: *e.g. bpsk*, *fsk*, *etc.*;
- an **interference** model: *e.g. CDMA*, *FDMA*, *etc.*.

Upon packet emission, by an internal node or an external program, WSNet generates two events:

- a *carrier sense* event that occurs at transmission start;
- a *packet reception* event that occurs at transmission end.

The carrier sense event is used to notify all nodes with the apparition of a new signal. This event provides each node with the radio resource (frequency, code, modulation, *etc.*), the SiNR (*Signal over Noise Ratio*), the reception power, and the BER (*Bit Error Rate*) of the signal computed for the considered node. This event enables the radio to lock on the packet when its transmission starts with the knowledge of the signal strength and the SiNR. With this event, one may easily simulate capture effects [9], loss of a signal, *etc.*

The packet reception event notifies the reception of the packet. Given a receiving node, the SiNR, the reception power as well as the BER are computed for each bit of the packet. This information is provided to the receiving node. Depending on the user configuration, the simulator either drops the packet as a function of the packet PER (*Packet Error Rate*) or introduces errors in the packet according to the BER.

5.2 SiNR computation

To compute the reception power of a packet, several factors are considered:

1. The transmission power;
2. The radiation pattern of the emitter's antenna;
3. The propagation model to compute the signal power at the receiver's location;
4. The radiation pattern of the receiver's antenna;
5. The sensibility of the receiver's antenna.

To compute the SiNR at a given time, for each byte of the considered packet, we consider the following factors:

1. The white noise of the receiver's antenna;
2. For each packet that is being transmitted during the considered byte transmission,
   (a) The reception power of the colliding packet;
   (b) The interference model and the radio resources that are used by the considered and the colliding packets.

Once the SiNR is known, the corresponding BER is computed using the modulation model. This radio medium simulation is very precise as the SiNR and BER are computed independently for each byte of the packet.

5.3 Node modeling

A WSNet internal node is modeled using the following blocks:

- a **mobility** pattern: *e.g. random waypoint, etc.*;
- an **application**: *e.g. CBR, Hello protocol, etc.*;
- a **queue**: *e.g. FIFO, priority queue, etc.*;
- a **MAC** protocol: *e.g. 802.11 DCF, S-MAC, etc.*;
- a **radio**: *e.g. half-duplex, full-duplex, etc.*;
- an **antenna**: *e.g. omnidirectionnal, directionnal, etc.*;
- a **battery**: *e.g. linearly decreasing etc.*

For each of these blocks, the model as well as the model parameters (*e.g. antenna white noise, sensibility*) are node specific.

WSNet nodes may also be simulated by external programs. Through a UDP/IP front-end, external programs may introduce packets in the WSNet radio medium simulator. WSNet computes the radio propagation, collisions, introduce errors according to the computed BER and returns the received packets to the external programs using UDP/IP communications. This design allows the setup of large distributed simulations using very accurate platform simulators like WSim. This is more detailed in section 6.

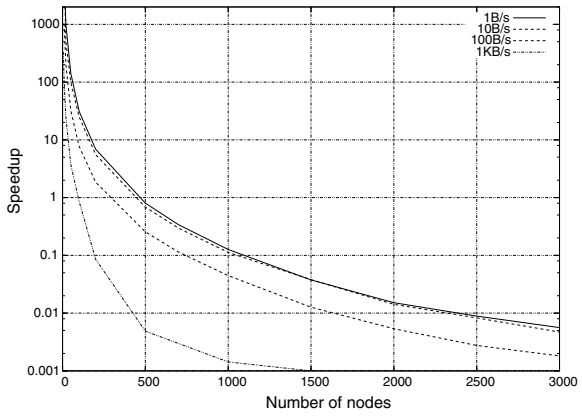

Figure 6: WSNet speedup as a function of the number of nodes and for various bandwidth usage.

5.4 Scalability

As an overview of WSNet scalability, Figure 6 gives the simulation speedup - simulated time over simulation time - for a varying number of nodes and a varying usage of the radio medium. These simulations have been performed with a Pentium IV at 2GHz and with 1GB RAM. The propagation model implements the Friis free space model with a pathloss exponent of 2, *i.e.* the reception power decreases proportionally to the distance at a power of two; we consider additive interferences [11]; the mac protocol is the 802.11 DCF and the bandwidth usage of each node varies from $1B/s$ to $1KB/s$. In these experiments, all nodes are in range of each others and thus neighbors regarding communications. In consequence, the density, in terms of communication neighborhood, becomes really high as the number of nodes increases.

As we can see, the speedup is highly dependent both on the number of nodes and the radio medium use. The node cardinality dependency is due to the computation of the packet reception at each node and the node radio medium use dependency is due to the fact that the simulator computes a reception power and SiNR for each byte transmitted in the network. The price of this accuracy in the radio medium simulation is a lower speedup than the one that would be achieved without consideration of interferences or with consideration only at packet scale.

Though the speedup decreases until 0.001 for 1000 nodes and a bandwidth usage of $1KB/s$, the simulator is still efficient regarding its accuracy. Moreover, it is very rare to have network topologies where the communication neighborhood is as high as 1000 nodes and in more conventional topologies with for example a neighborhood of 50 nodes, the speedup remains high. These speedup values are also more than competitive compared to the ones achieved by classical network simulators as presented in [6].

6. SIMULATION USING WSim AND WSNet

Using the WSNet UDP/IP front-end, WSNet and WSim can be used together to form the Worldsens simulation platform which offers the opportunity to simulate with high accuracy a whole sensor network. Each node of the sensor network is simulated by a WSim program while the radio medium simulation is performed by WSNet. The simulation platform is depicted in Figure 7. In this context, all features of WSim and WSNet are available. For example, it is possible to use several serial connections or gdb connections to different WSim programs while simulating a whole sensor network. In this last case all the nodes are stopped and the complete simulation is put in step-by-step mode according to the nodes that are debugged. This allows to use the Worldsens environment to debug distributed protocols at the source level using only simulation tools. All models of WSNet are also available, enabling a precise radio medium simulation and making Worldsens differ from other simulation tools like Tossim [14] or Avrora [16] where only very simple radio medium models are offered.

Each node of the network is simulated by WSim which takes as input the binary file of the application. As described in section 4, the simulator runs the native code as deployed in the sensor hardware without any change and emulates all components embedded in the hardware sensor nodes: *e.g.* the flash memory module, the DS2411 1-Wire silicon serial number and CC1100 multi-channel RF transceiver, *etc.* Thus, all instructions sending commands to the CC1100 are executed and the behavior of the CC1100 is also simulated. When the CC1100 actually transmits a byte, it is transferred to the radio medium simulator (*i.e.* WSNet) which simulates the radio propagation and interferences according to its internal models, and finally transmits the data to the simulated CC1100 RF transceivers in the other WSim programs. At the network level, data transmission is resolved with a one byte granularity.

6.1 Simulators synchronization

As the global simulation platform relies on different programs and thus different processes, no global time is available. In a network simulation, node asynchronism has no impact as long as the nodes do not interact. However, as soon as sensors exchange packets, the different simulated nodes must be synchronized. This is crucial for the evaluation of real-time applications or the good functioning of time-constrained protocols like for example MAC protocols with sleep and activity periods (*e.g. S-MAC*). In other words, when a packet is transmitted, the simulated time since the last synchronization must be equal in all simulated nodes. To perform synchronization, we use a save/backtrack mechanism together with *rendezvous* points (RPs). WSNet is responsible for setting RPs during the simulation. RPs are not only set at packet reception but also periodically in order to regularly synchronize the network and avoid long backtracks. These periodical RPs frequency is set *priori* and can be tuned empirically offline depending on the simulated application.

When a RP is set, WSNet notifies the WSim programs of the next synchronization point. A WSim which has not reached this point yet simulates until the RP, notifies WSNet and then blocks waiting for a release from WSNet. A WSim already ahead of the RP backtracks to the previous RP and re-simulates until reaching the new RP. Then, it notifies WSNet and blocks waiting for a release from WSNet. When all WSim have reached the RP, WSNet solves the eventual event which has caused the RP, *i.e.* data transmission, sets a new RP and releases the WSim programs. When released, a WSim saves its internal state. This internal state backup is eventually used during a future backtrack.

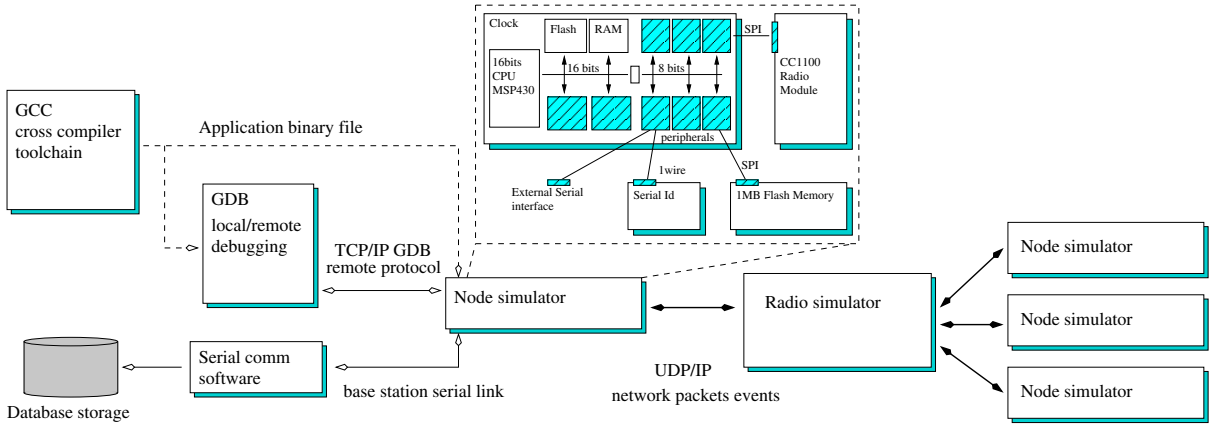

Figure 7: Distributed simulation environment for Wireless Sensor Networks.

6.2 Distributed simulation and scalability

As the communication between WSim and WSNet relies on multicast UDP/IP and as no global clock is required, it is possible to distribute a simulation over a cluster of workstations. This feature allows the Worldsens simulation framework to scale well up to a large number of simulated nodes, depending on the number of available workstations. Regarding its scalability capacity, Worldsens differs from other approaching simulation tools like Avrora which simulates all the nodes in a single process.

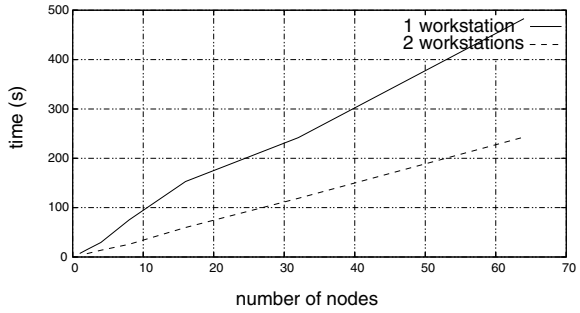

Figure 8: Scalability of distributed mixte simulations using WSNet and WSim with regards to the number of nodes. (Simulated time is 5 seconds)

Figure 8 shows the distributed simulation time taken for different number of nodes in the sensor network. During this experiment, each node is simulated for a 5 seconds period of activity during which it periodically broadcasts a `Hello` packet and listens for the medium the rest of the time. Each emitted byte lead to a RP and thus potential backtracks. In consequence, in this simulation where each node transmits packets, the number of RPs increases at least linearly and potentially more with the number of nodes.

In a first step, all sensor nodes and the wsnet process were run on a single machine using the local loopback as the interconnection network. Experiments were done on a 1.4GHz pentium mobile laptop with 768MB of RAM running Linux. This low-end machine allows to simulate up to 64 sensor nodes with a linear speedup with regards to the number of nodes. In a second step, the same simulations were performed using two machines, the previous one together with an AMD Sempron 2GHz with 1GB of RAM and using a 802.11g network for communications. As we can see, the simulation time is still linear with regards to the number of simulated nodes. However, this simulation time is twice shorter than the one obtained using a single machine. This gain is due to the intrinsic architecture of Worldsens where the load of the WSim simulators can be balanced over the different workstations. In terms of scalability, it represents a clear advantage compared to solutions where all nodes are simulated in a single process.

The scalability of the simulation scheme is highly dependent on the number of nodes but also on the communication activity of the simulated nodes. However, it is hard to clearly analyze the impact of this last parameter. An increase in the number of communications leads to an increase in the number of RPs and potential backtracks. However, these backtracks take less time as the delay between two synchronization points, *i.e.* two communications, is reduced. Nodes are more synchronized and the time lost for synchronization is counter-balanced by the backtrack cost reduction. Nevertheless, time required by wsnet to resolve radio receptions does increase with the number of exchanged information.

A last issue concerns reliability of the synchronization protocol. If simulations are distributed over a higher number of workstations communicating over an unreliable media, we can expect to lose synchronization packets. The use of a reliable protocol is an on-going work that would provide the robustness required for the set-up of large scale distributed simulations.

7. A CASE STUDY

To demonstrate the use of the Worldsens development framework, we present here some of the steps which have taken place during the development of one particular sensor network application for experimentations in our university. Within the context of a university-led project and to study large graphs of interactions, it has been planned to distribute sensor nodes to more than 200 students of a given department. The task of these sensor nodes is to periodically discover and log their neighborhood, and to download the gathered data to base stations disseminated in the university. The collected data will be useful to study graphs of interactions and students mobility. To reach this objective,

it has been necessary to tune the hello protocol in order to minimize the energy consumption for long-time experimentations and to be reactive enough to detect the neighboring students up to a given distance and despite mobility.

7.1 Example of WSNet use

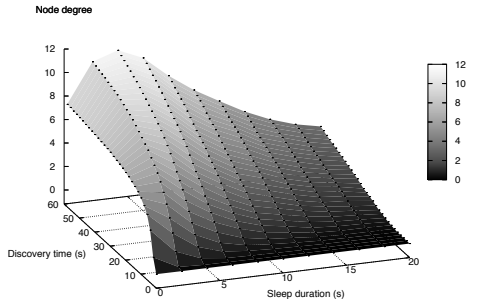

Figure 9: Node degree for $T_s = 0...20s$ and $T_w = 2s$.

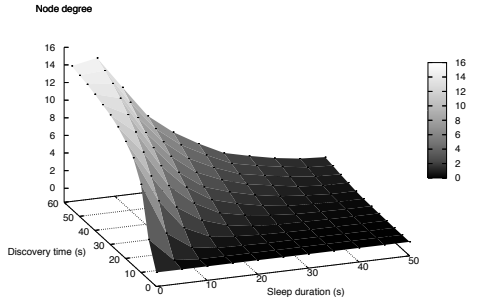

Figure 10: Node degree for $T_s = 0...50s$ and $T_w = 5s$.

During one of the first steps of the application design, WSNet has been used to dimension the hello protocol used to discover one sensor node's neighborhood. In this protocol, each node can be in one of the following three states: listening, talking or sleeping. These three states are performed inside a frame, F, of size $T_w + T_s$. For each occurrence of the frame F, a node transmits only one hello message with a duration of δ, listens to the medium during $T_w - \delta$ and sleeps during T_s.

As nodes sometimes sleep, neighbors are not always discovered. Moreover, some hello packets may also not be received because of collisions. Increasing T_s leads to an increase in the probability that a neighbor is sleeping during a given node emission and thus may lead to a decrease in the number of discovered neighbors. On the opposite side, it also induces a decrease in the collision probability and thus may lead to a potential increase in the number of discovered neighbors. There is a clear trade-off and the use of WSNet offers the opportunity to choose the best parameters regarding our constraints.

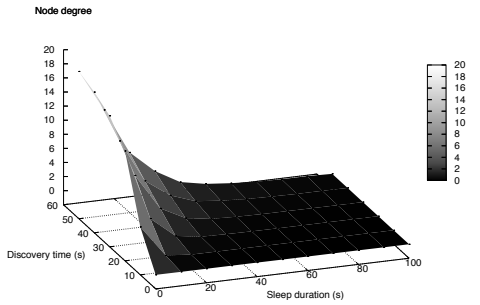

Figure 11: Node degree for $T_s = 0...100s$ and $T_w = 10s$.

Given the desired physical medium characteristics and the prevailed node density, we have used WSNet to compute the evolution of a sensor node degree. As an illustration, Figures 9, 10 and 11 show a sensor node degree for various values of T_s and a value of T_w respectively equal to 2s, 5s and 10s. These simulations were performed using a path-loss propagation model, additive interferences and a step function for modulation. [2] describes in more details the tuning of this hello protocol.

7.2 Example of WSim use

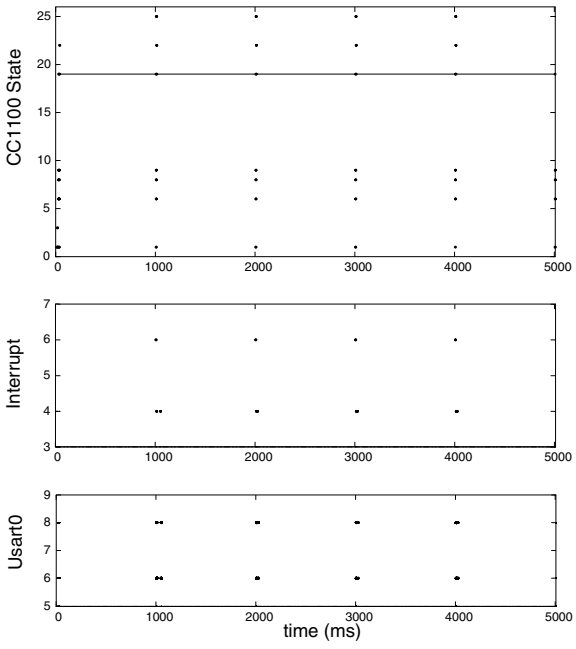

Figure 12: Some of the WSim traces for the hello application.

After implementation of the application, the target binary code has been simulated using WSim and WSNet. Among the traces, Figure 12 shows the Chipcon CC1100 R/F transceiver states (as defined in the datasheet) of one

node during the simulation of a sensor network. The sensor node sends a hello packet each second. The lower part of the figure shows the interrupt triggered by the application every second to send a packet and the interrupt triggered by the CC1100 to notify the transmission termination. The middle part of the figure shows the data enter and exit time from the serial port in SPI mode, including the data shift register. Finally, the upper part of the figure corresponds to the different states taken by the transceiver during the simulation (states numbers are taken from the CC1100 technical datasheet).

For example, the use of WSim in this distributed simulation with real trafic allows to perform implementation choices such as choosing between synchronous (active polling) or asynchronous (interrupt driven) communications between the microcontroller and the transceiver to feed its Tx fifo.

8. CONCLUSION

In this article, we have presented Worldsens, an integrated platform for the design, development, prototyping and deployment of applications for wireless sensor networks. This integrated platform consists in two simulators which simulate complete sensor networks with a very high accuracy. As an example, WSim implements all the micro-controller clock manager specifications and allows to perform clock drift during the simulation on one given node or one specific clock of a given node. WSim is more accurate than existing simulators like Atemu and Avrora but still offers scalability and performances. On a single machine, the scalability is linear in the number of simulated nodes. Worldsens also offers powerful debugging and profiling tools. These tools gives the opportunity to analyze a real code application in terms of performances but also regarding energy consumption. These tools are mandatory for the profiling of hardly constrained applications.

The availability of very accurate simulators offers a great ease of use and the opportunity to tune a code without the burden of deploying the application on small devices with limited interface capacities.

We are now interested in performing a more comprehensive study of the energy consumption. We are also working on code profiling at the source code level for embedded softwares. We want to propose some domain specific language (DSL) approaches adapted to low level sensor network application parts like for example the MAC layer. Indeed, given a specific application, the MAC layer should be tuned in order to offer a good behavior in terms of energy, latency, bandwidth. Developing a MAC layer is generally a tedious part. Combining the use of DSL and the Worldsens platform may offer a flexible way to optimize the low layers of a sensor network application.

Although several models are already present in WSNet, we expect to implement and offer additional ones (MAC, radio propagation, radio modulation). We are also working on a extension of WSNet to directly bind the simulator to external programs such as accurate propagation tools that take into account an indoor environment as for example [12] or to enable the upload of real mobility traces gathered during real experimentations.

9. REFERENCES

[1] M. Ali, U. Saif, A. Dunkels, T. Voigt, K. Rmer, K. Langendoen, J. Polastre, and Z. Uzmi. Medium access control issues in sensor networks. *ACM Computer Communication Review*, 36(2):33–36, 2006.

[2] E. Ben Hamida, G. Chelius, and E. Fleury. Revisiting neighbor discovery with interferences consideration,. In *Third International Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks (PE-WASUN)*, Torremolinos, Spain, October 2006. ACM.

[3] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han. MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms. *MONET*, 2005.

[4] G. Chelius, A. Fraboulet, and E. Fleury. Worldsens: a fast and accurate development framework for sensor network applications. In *The 22nd Annual ACM Symposium on Applied Computing (SAC 2007)*, Seoul, Korea, March 2007. ACM.

[5] E. Egea-Lopez and *al.* Simulation tools for wireless sensor networks. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, USA, July 2005.

[6] E. Egea-Lopez, J. Vales-Alonso, A. Martinez-Sala, P. Pavon-Mariño, and J. Garcia-Haro. Simulation scalability issues in wireless sensor networks. *IEEE Communications Magazine*, pages 64–73, July 2006.

[7] Alois Ferscha. *Parallel and Distributed Computing Handbook*, pages 1003 – 1041. McGraw-Hill, 1996.

[8] H.T. Friis. A note on a simple transmission formula. *In Proceedings of IRE*, 1946.

[9] S. Ganu, K. Ramachandran, M. Gruteser, and I. Seskar. Methods for restoring mac layer fairness in ieee 802.11 networks with physical layer capture. In *Proceedings of REALMAN 2006 in conjunction with ACM MobiHoc 2006*, Italy, May 2006. ACM Press.

[10] Lewis Girod, Thanos Stathopoulos, Nithya Ramanathan, Jeremy Elson, Deborah Estrin, Eric Osterweil, and Tom Schoellhammer. A system for simulation, emulation, and deployment of heterogeneous sensor networks. In *2nd international conference on Embedded networked sensor systems*. ACM Press, November 2004.

[11] P. Gupta and P. Kumar. Capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2):388–404, 2000.

[12] E. Jullo J.-M. Gorce and K. Runser. An adaptative multi-resolution algorithm for 2d simulations of indoor propagation. In *12th IEE Int. Conf. on Antennas and Propagation*, 2003. Best paper award on Propagation.

[13] Available online, http://kcachegrind.sourceforge.net/, nov 2006.

[14] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. Tossim: accurate and scalable simulation of entire tinyos applications. In *SenSys'03*, 2003.

[15] J. Polley, D. Blazakis, J. McGee, D. Rusk, and J. Baras. ATEMU: A fine-grained sensor network simulator. In IEEE, editor, *Sensor and Ad Hoc Communications and Networks*, 2004.

[16] Ben L. Titzer, Daniel K. Lee, and Jens Palsberg. Avrora: scalable sensor network simulation with precise timing. In *IPSN '05*, 2005.