Un Modelo de Simulación para Redes de Sensores Inalámbricas Basado en TOSSIM

Francisco Javier Rincón Vallejos

Proyecto final de Máster

Dirigido por: Marcos Sánchez-Élez Martín y David Atienza Alonso

Universidad Complutense de Madrid

Departamento de Arquitectura de Computadores y Automática

Máster en Investigación en Informática

June 2007

Contents

Abstract

Wireless Sensor Networks (WSNs) are composed of a large number of very tiny devices used for biomedical or environmental monitoring applications. These devices cooperate among themselves and allow us to observe plenty of natural processes. In this work we propose an accurate model based on TOSSIM for wireless sensor devices to be able to do simulations in order to estimate network performance and energy consumption. This model helps us to evaluate the impact on the network performance of making changes in the node architecture or in the communication layers. We have tested our model with the Sensor Cube nodes, developed by IMEC. In order to show the efficiency of our model we have simulated the energy consumption for different MAC protocols.

Keywords

Wireless Sensor Network, MAC protocol, Operating System.

Abstract (Spanish)

Las Redes de Sensores Inalámbricas se componen de un gran número de dispositivos diminutos que se usan en aplicaciones de monitorización biomédica o ambiental. Estos dispositivos cooperan entre sí y nos permiten observar multitud de procesos naturales. En este trabajo, proponemos un modelo preciso basado en TOSSIM para dispositivos sensoriales inalámbricos que nos permita hacer simulaciones para estimar características de la red como pueden ser el rendimiento o el consumo de energía. Este modelo nos ayuda a evaluar qué impacto tienen en el rendimiento de la red cambios en la arquitectura del nodo o en las capas de comunicación. Hemos probado nuestro modelo con unos nodos denominados Sensor Cubes, desarrollados en el IMEC. Para mostrar la eficiencia de nuestro modelo hemos realizado simulaciones para obtener el consumo de energía con distintos protocolos MAC.

Palabras clave

Red de Sensores Inalámbrica, protocolo MAC, Sistema Operativo.

Chapter 1
Project Overview

1.1 Motivation

Wireless Sensor Networks [1] (WSN) are wireless networks consisting of spatially distributed autonomous sensors to cooperatively monitor physic magnitudes. Their utility ranges from environmental measurements at different locations to medical monitoring. The wireless sensor network main workload entails sensing relevant information by the nodes, collecting that information, interpreting it, recording it and communicating it to a base station. In order to perform those activities each sensor node is made up for some sensors, a radio transceiver and a microcontroller. As power source we usually have a battery, but energy harvesting modules (like solar cells or thermoelectric converters) can also be included, increasing the life of the node and therefore minimizing its maintenance [2].

The type of sensors that can be used in a node is not restricted, so WSNs have unlimited applications. It can be used, for instance, temperature, humidity or illumination sensors for environmental monitoring [3], ECG/EEG (electrocardiogram/electroencephalogram) sensors for healthcare applications [4], activity sensors for home automation, etc. We can also use these networks for traffic control, seismic and fire detection, sports, etc.

The applications for WSNs are many and varied. They are used in commercial and industrial applications to monitor data that would be difficult or expensive to monitor using wired sensors. They could be deployed in wilderness areas, where they would

remain for many years (monitoring some environmental variables) without the need to recharge/replace their power supplies.

Each node has a microcontroller which provides computational power and memory. With this microcontroller the node is able to process the data read by the sensors before sending it, allowing consequently to decentralize the network. Using this distributed processing, the communication between nodes, and therefore the energy consumption, is significantly reduced. With this approach, big storage units or powerful processing elements to store and process all the data collected by the nodes are not needed, since not all the information is sent to the base station, but only the relevant part after processing it.

Other main component is the radio chip, which provides the nodes the ability to communicate to each others. Using a radio chip instead of wires to communicate the nodes among themselves we achieve mobility, one of the major properties of WSN, allowing the utilization of WSN over any environment. Manufacturers are developing new micro-radios with lower energy consumption and adapting them to the needs of sensor networks [5]. Some of these chips include MAC protocols like Bluetooth [6], ZigBee [7], etc. performed by hardware.

Depending on the application, plenty of parameters can be changed to adapt the network to it. For example, we can adapt the network topology to improve one particular monitoring characteristic; or we can use different MAC [8] and routing protocols centered in reduce energy consumption. We also can consider the possibility to transmit internal information of the node to its neighbors, for instance, to inform them about low battery charge. Other interesting property of these networks is that they can change dynamically the application that is being executed in a concrete moment. If we have a node equipped with temperature and humidity sensors, perhaps in some moments we are interested only in the temperature and in other moments we are more interested in the humidity, we can change the application that is running in the nodes to adapt it to our interests.

In order to measure the impact in the performance of changes in the configurable parameters of the network and estimate energy consumption, reliable simulators and

therefore reliable models of sensors architectures are highly needed. We can make changes in the WSN architecture model for simulation before making them in the real platform. For example, if we require to test a new MAC protocol, it can be implemented, debugged and tested to get values of its impact in both performance and energy consumption, before implementing it in the real node.

1.2 Report Structure

Up to now, we have showed the motivations to propose a model to be able to run very reliable and accurate simulations. These simulations allow us to study changes in the node architecture before actually doing them. The remainder of this chapter provides information about the different applications of WSN. The rest of the report is organized as described below:

- Chapter 2 provides a description of the hardware used to validate the proposed model and briefly describes some other platforms for WSN.
- In chapter 3 we present an overview of the operating system that is used in the nodes and its simulator. Some other operating systems and simulator are also discussed.
- Chapter 4 presents the energy model that we propose.
- Chapter 5 analyzes the energy model with a case study.
- Chapter 6 gives an idea of the future work.
- In chapter 7 several conclusions are discussed.

1.3 Applications

The table below shows some applications scenarios and a few examples for WSN that are envisaged by researchers in different sectors:

Table 1.1. Applications of Wireless Sensor Networks.

Area	Examples
Biologic research	Non-invasive habitat monitoring and study of wildlife species/populations.
Environmental monitoring	Pollution (air, water, soil); early alarms for forest fires, volcanoes, earthquakes, and other natural hazards.
Disaster response	Emergency support for disaster recovery scenarios to identify risks and hazards, locate people/survivors.
Healthcare	Staff and patient tracking; on-the-body vital signs monitoring; drug administration control.
Agriculture	Livestock and crops monitoring; microclimate management for dairy production or wines; soil fertility analysis.
Industrial sector	Manufacture process automation, monitoring & control; equipment failure prediction; production quality assurance.
Retail sector	Stock management, product tracking, quality monitoring.
Architecture	Office smart spaces; home automation; intrusion detection.
Transportation	Monitoring of internal systems in cars, ships and aircrafts.
Local authorities	Road traffic reporting, analysis and coordination.
Military	Surveillance; enemy identification; target acquisition and tracking; logistic operations support.

1.4 Related work

There are some well known groups working in the field of WSN. One of the most active groups belongs to University of California, Berkeley. Apart from designing some very extended platforms like Mica [9] or Telos [10] motes, they have carried out some other well know projects like the development of TinyOS [11] (an operating system for WSN) and TOSSIM [12] (a WSN simulator for TinyOS). NesC [13] programming language was also developed by this group. They also developed the SmartDust [14], a hypothetical network of tiny wireless microelectromechanical systems (MEMS) sensors, robots, or devices, installed with wireless communications, that can detect anything from light and temperature, to vibrations, etc. Currently, they are working in another project called NEST [15], an Open Experimental software/hardware Platform for

Network Embedded Systems Technology research that will accelerate the development of algorithms, services, and their composition into challenging applications dramatically.

There are some other groups that have designed and developed other sensor platforms like BTNode [16] (in the Swiss Federal Institute of Technology, Zurich), DSYS25 [17] (at University College Cork, Ireland) or ESB [18] (at Freie University, Berlin).

There exist some applications in different fields that have been implemented using WSN. One example is ZebraNet [19], a project developed at Princeton University, whose goals are to develop, evaluate, implement, and test systems that integrate computing, wireless communication, and non-volatile storage along with global positioning systems (GPS) and other sensors. On the biology side, the goal of ZebraNet is to use systems to perform novel studies of animal migrations and inter-species interactions.

Other application is a habitat monitoring system on Great Duck Island, off the coast of Maine. Researchers deployed a 35-node network on the island to monitor the presence of Leach's Storm Petrels in their underground burrows [20]. The network was designed to run unattended for at least one field season (7–9 months). Nodes, placed in burrows, monitored light, temperature, relative humidity, pressure, and passive infrared; the network relayed readings back to a base station with an Internet connection via satellite, to be uploaded to a database.

CodeBlue [21] is other example, it is under development at Harvard University. This application consists in a set of devices that collect heart rate (HR), oxygen saturation (SpO2), and EKG data and relay it over a short-range (100m) wireless network to any number of receiving devices, including PDAs, laptops, or ambulance-based terminals. The data can be displayed in real time and integrated into the developing pre-hospital patient care record. The sensor devices themselves can be programmed to process the vital sign data, for example, to raise an alert condition when vital signs fall outside of normal parameters. Any adverse change in patient status can then be signaled to a nearby EMT or paramedic. The device used has been also developed by this group, and consists in a wearable wireless pulse oximeter and 2-lead EKG based on the Mica2, MicaZ, and Telos sensor node platforms

1.5 Acknowledgements

The author would like to thank the following people:

- *Alexandru Susu* and *Andrea Acquaviva* for helping me to begin with this research.
- *Nadia Khaled* for helping me with the radio communications.
- *Holst* for providing the necessary hardware for this research.
- *Tom Torfs* (from IMEC) and *Julien Penders* (from Holst) for their feedback to my questions on understanding the hardware sensor platform.
- *Michele Paselli* (from Holst) for helping me to port the TDMA protocols to the Sensor Cubes and for the new structure of the drivers.
- *Héctor Sanjuán, Nora Pacheco, Rubén Braojos* and *Víctor Alaminos* for porting two applications for ECG monitoring to the Sensor Cubes.
- My tutors: *Marcos Sánchez-Élez* and *David Atienza* for guiding me during the Master.
- *Giovanni de Micheli* for cooperating with my tutors in guiding me, and for inviting me to his department at EPFL.
- My family and Laura.

Chapter 2
Sensor Cubes

In order to validate our model we use a platform called Sensor Cube, developed by IMEC [22]. The total size of the platform is 14x14x18mm^3. The Sensor Cube is depicted in figure 2.1, placed on a 2-euro coin. Figure 2.3 shows an alternative implementation of the node, even smaller, using solder ball interconnect technology.

One of the major characteristics of this platform is its modular design. The Sensor Cube is composed of different blocks of hardware components, that can be easily plugged together to form sensors with different capabilities every time. The Sensor Cube is divided in different layers with different functionality. The top layer is responsible for the wireless communication. It consists on a single-chip short-range 2.4GHz transceiver (Nordic nRF2401, 18 nJ/bit) with a coplanar integrated folded dipole antenna [23].

The next layer controls the entire module, an MSP430 microcontroller fulfils this task [24]. The microcontroller has a very low active power (0.6 nJ/instruction), low standby power (2μW), fast wakeup from standby to active mode (6μs) and on-chip 12-bit analog-to-digital converter. A 32.768kHz ±20ppm crystal provides the system's local time reference.

A power management layer has been implemented, which accepts power from an energy scavenger such as a compact solar cell or thermoelectric generator, and uses it for keeping a secondary battery charged.

The final component of the sensor node is the sensor layer which measures different parameters, depending on the kind of sensors: temperature, humidity or illumination for

environmental monitoring or electrocardiogram, electroencephalogram, etc. for medical applications.

IMEC has also developed a USB stick-like radio receiver, depicted in figure 2.2. It is connected to a PC and receives the data from all the sensor nodes in the network. This USB stick has the same architecture as the nodes with the difference of not having neither sensors (because it is used as a base station), nor power management system, since it takes energy from the PC which is connected to.

Figure 2.1. Sensor Cube on 2-euro coin

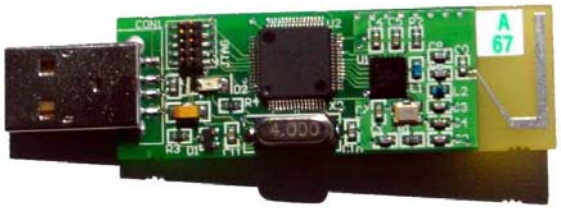

Figure 2.2. USB stick

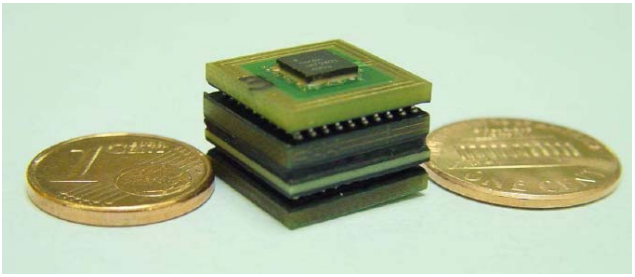

Figure 2.3. Alternative hardware implementation using solder ball interconnect technology.

In the following subsections, the different layers of the IMEC Sensor Cube are considered and some key features of the hardware are presented. By understanding the underlying platform and its capabilities, someone could understand better the limitations, ideas and solutions presented in later chapters.

2.2 Texas Instruments MSP430 Microcontroller

The Microcontroller Unit (MCU) selected by IMEC to equip the Sensor Cubes is the ultra low power MSP430F149 from Texas Instruments. This microcontroller consists of several devices featuring different sets of peripherals targeted for various applications. The architecture, combined with five low power modes is optimized to achieve extended battery life in portable measurement applications. The device features a powerful 16-bit RISC CPU, 16-bit registers, and constant generators that attribute to maximum code efficiency. The digitally controlled oscillator (DCO) allows wake-up from low-power modes to active mode in less than 6 µs. MSP430 architecture is depicted in figure 2.4.

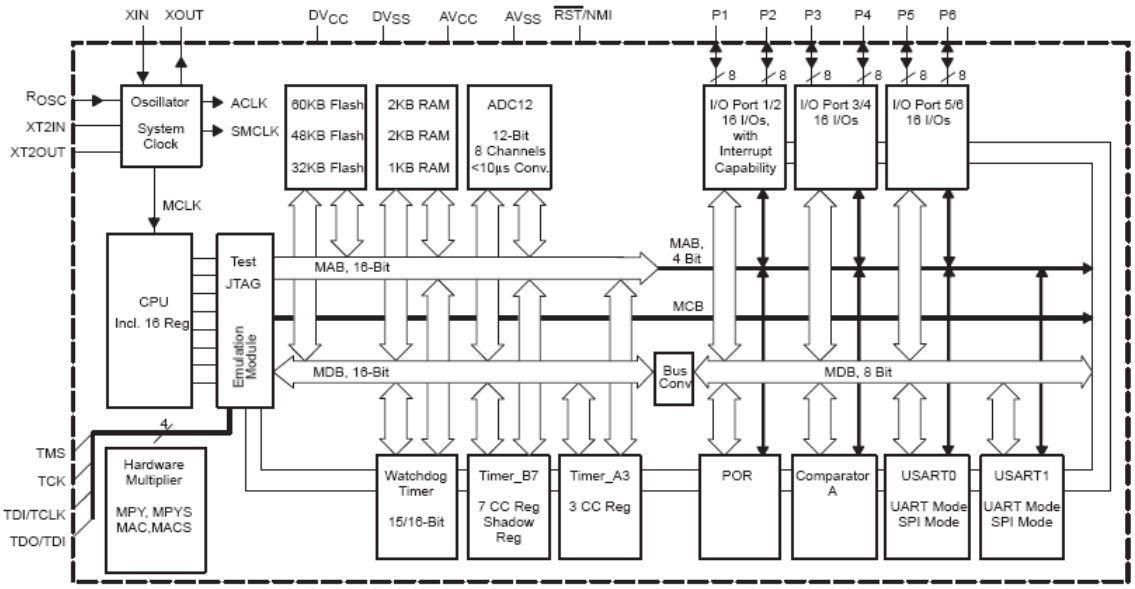

Figure 2.4. MSP430 architecture

The Texas Instruments MSP430F149 has two built-in 16-bit timers, a fast 12-bit A/D converter, two universal serial synchronous/asynchronous communication interfaces (USART), and a total of 48 I/O pins for connecting peripheral components.

Regarding the memory of the system, a total of 2KB of RAM is available, backed by (a large for the microcosm of sensors) 60KB of Flash/ROM. Both memories can be used for code and data. In the case of Flash/ROM, word or byte tables can be stored and used directly with no need for them to be copied to RAM.

Typical applications include sensor systems that capture analog signals, convert them to digital values, and process and transmit the data to a host system. The timers make the configurations ideal for industrial control applications such as ripple counters, digital motor control, EE-meters, hand-held meters, etc. The hardware multiplier enhances the performance and offers a broad code and hardware-compatible family solution. Now, we are going to describe in more detail the different components included in the MSP430.

2.2.1 Digital I/O of the MSP430 MCU

The MSP430 MCU used in the IMEC platform has a total of 48 I/O pins, grouped in a total of 6 I/O ports (P1 - P6) of 8 pins each. Throughout this document individual pins will be identified by their port number, followed by their “in-port” number (for example, the first pin of the first port P1 would be P1.0 and the last P1.7). Users can configure individual pins to have input or output direction and can also read and write to them at will. Two out of the six ports of the MCU (ports P1 and P2) offer interrupt capability for each and every pin involved (i.e. from pin P1.0 to P2.7) and can be configured to interrupt the CPU on a rising or falling edge of an incoming signal.

The configuration of the digital I/O is done by the user and typically a C header file contains the desired pin settings. By setting values of dedicated “I/O setup registers”, the user can select the initial value of the pins (0 or 1) and the direction of the pins (INPUT or OUTPUT) depending on the way that they will be used. In addition to the above, the functionality of individual pins can be selected to be simple I/O or to provide peripheral functionality of the MCU like SPI explained below. The correct

configuration of the I/O pins is vital for the successful communication of data to and from the MCU and for its operation in general.

2.2.2 Universal Synchronous/Asynchronous Receive/Transmit

The Universal Synchronous/Asynchronous Receive/Transmit (USART) is an interface to a single hardware module which supports two serial modes of communicating data to and from peripheral devices connected to the MCU. The modes available that are of interest to us are the Universal Asynchronous Receive/Transmit (UART) and the Synchronous Peripheral Interface (SPI). The MCU used in the IMEC Sensor Cube provides two USART modules with the same functionality: USART0 and USART1.

The synchronous (SPI) mode allows the MCU to connect to a peripheral device (e.g. Radio Chip) as either a master or a slave, depending on which device controls the communication. When using this mode, three pins of the MCU are of interest: Slave In Master Out (SIMO), Slave Out Master In (SOMI) and USART Clock (UCLK). Consider a set up where the MCU is the master and a Radio Chip is the slave. The SIMO pin is used when the master sends data to the slave (MCU -> Radio) and the SOMI pin is used when the slave has data to send to the master (Radio -> MCU). The UCLK is used as a clock source that synchronizes and controls the data rate of the transfers.

The communications through SPI are Byte oriented and utilize independent transmit and receive shift registers and buffers. In the set up described above, when the MCU wishes to write a byte to the Radio, it does so by writing it bit by bit on its transmit shift register. While this occurs at the rising edges of the UCLK, the receive shift register of the MCU is filled bit by bit with data from the transmit register of the slave-Radio concurrently (but at the falling edges of the clock). It is therefore a requirement to write something to a slave in order to get the data it is willing to transmit to the master.

An alternative way of communicating data via a connection is to do so without using a hardware controller like SPI. Such methods are known as bit banging techniques and mainly involve simulation of a protocol in software. The MSP430 allows the use of

such techniques and, therefore, its general purpose pins can be used for the emulation of serial communication interfaces.

2.2.3 Clock System and Timers

The clock system of the MSP430 uses a low frequency clock input driven from a common 32kHz watch crystal. This power efficient clock source is used to provide three basic timer modules, namely, the Watchdog Timer, Timer_A and Timer_B. The watchdog timer module has the sole purpose of initiating a system restart after a serious software error. In the case that the watchdog function is not needed, the timer can be used to "fire" (generate interrupts) at predetermined intervals. Both Timer_A and Timer_B are similar 16-bit timers/counters with the capabilities of a) capturing the time of an event occurrence (signal or interrupt) and of b) generating an interrupt at predetermined intervals. Both timers are also capable of generating interrupts on counter overflow. The timers can operate in various modes as they can be configured to count upwards (0-65535) downwards (65535 - 0) Up-Down (0-65535-0) and Continuously. Their clock source that ultimately controls their tick granularity and therefore their precision, can be selected and configured at will.

An interesting feature available by both Timer_A and Timer_B is the ability to use them in Capture Mode. The Capture Mode is used when there is a need to record the time that an event has occurred. In order to do so, the capture inputs of the timer are associated with the particular event of interest (an MCU pin or an internal signal). The user can register interest on rising (0 to 1), falling (1 to 0) or both transitions of the bespoken pin or signal value. Upon capture, the time will be stored in a buffer and an interrupt will be raised by the timer. The capture feature of timers is very useful in situations where an interrupt must be raised when a pin value changes, but the pin happens to be in a port that without interrupt capability. With the above state of affairs, a neat way of "extracting" an interrupt is by associating a capture timer with the pin of interest. Now, upon transition, the timer which is monitoring the pin, will capture the time of the transition occurrence, but at the same time will raise the timer interrupt providing thus the required CPU interruption.

2.2.4 ADC12 Analog-to-Digital Converter

The ADC12 is an accurate 12-bit Analog-to-Digital converter and is onboard the MSP430 MCU. It is a high performance module capable of converting analog signals (e.g. from sensing hardware) to corresponding 12-bit digital representations. By employing a dedicated buffer, the ADC12 is capable of converting and storing samples without any CPU intervention and at the very high rate of 200 thousand samples/sec. For ease of use, conversions can be triggered by software or any of Timer_A or Timer_B. The signal to be digitized is fed to the ADC module from one of its eight input channels that are easily configurable. It is also worth noting that when the ADC12 is not actively converting, the core is automatically disabled and automatically re-enabled when needed, a fact that underlines again that the MSP430 was designed with low power consumption as a primary requirement.

2.3 Nordic nRF2401 Radio Transceiver

In order to develop an accurate simulation tool we need to study all the layers of the node. However, as the part of the nodes that consumes more energy is the radio chip (typically between 70% and 90% of the total energy consumed by the node, depending on the application), this is the component that we emphasize in our model.

The Radio Transceiver chip used in the Sensor Cubes is the nRF2401 from Nordic Semiconductors. The nRF2401 is a complete radio system incorporated onto a single piece of silicon. It uses 0.18μm CMOS technology and is designed for the 2.4GHz to 2.5GHz radio band. The three main aims for this product are cost, current consumption and simplicity of use, while maintaining good RF performance. The transceiver consists of a fully integrated frequency synthesizer, a power amplifier, a crystal oscillator and a modulator. Output power and frequency channels are easily programmable by use of the 3-wire serial interface. Current consumption is very low, only 10.5mA at an output power of -5dBm and 18mA in receive mode. Built-in Power Down modes makes power saving easily realizable.

This device provides a maximum data rate of 1Mbps and performs multi channel operation (125 channels). The channel switching time is less than 200μs and it supports

frequency hopping. The chip also performs address and CRC (Cyclic Redundancy Check) computation.

The following are the modes of operation that a Nordic nRF2401 may be and their related current consumption:

- Transmit Mode: 13mA (Average @ 0dBm output Power)
- Receive Mode: 23mA (Average for both channels ON @ 0dBm output Power)
- Configuration Mode : 12uA (Average)
- Stand-By Mode : 12uA (Average)
- Power Down Mode @ 400nA (Average)

The radio chip inserted in the Sensor Cubes has two active (RX/TX) modes of operation: Direct Mode and ShockBurst, which are described in the following subsections.

2.3.1 Direct Mode of operation

In direct mode the radio works like a traditional RF device. Data must be at 1Mbps ±200ppm, or 250Kbps ±200ppm at low data rate setting, for the receiver to detect the signals. This requirement dictates the employment of a clock source fast and accurate enough to enable the clocking of the data from the MCU into the Radio chip at those specified rates. In the case that transmission at 1Mbps is needed, then a 16MHz crystal is required. It should be made clear that unless the data is clocked into the chip at the correct rate, they will never reach their destination, since no receiver will pick-up the signal.

2.3.2 ShockBurst Mode of operation

The ShockBurst technology uses on-chip FIFO to clock in data at a low data rate and transmit at a very high rate thus enabling extremely power reduction. This mode of operation reduces the consumption. We can use a low frequency crystal to clock in data but the transmission rate is very high. Other advantage of this mode is the reduction of the processing in the microcontroller, because the radio chip also checks the address and

CRC of the receiving packet. In this operation mode, before sending a packet, the radio adds to it some information: preamble for the receiver to detect the incoming data (and recover the clock if we are using direct mode of operation), address and CRC. The receiving node checks all this information, if the packet address is not the same as the node address, i.e. the packet is not for that node, the packet is discarded; in the same way, the packet is discarded if the CRC check is not correct.

The structure of the ShockBurst packet is depicted in figure 2.5. The preamble is 8 bits long, the address can be from 8 to 40 bits long, the payload size is 256 bits minus the following: (Address: 8 to 40 bits + CRC: 8 or 16 bits). The CRC is optional and is 8 or 16 bits long. In our case we use 8 bits for the preamble, 24 for the address, 216 for the payload and 16 for the CRC.

Figure 2.5. Data package diagram

To justify even further this approach, consider a MCU that is only capable of providing data to the Radio chip at low data rates (e.g. 10kbps), and wishes to transmit a total of 100 bits. If the Radio wants to transmit immediately and as the data arrived from the MCU, the transmission time would be 10ms, forcing the Radio to stay in transmit mode for that time. More often than not, though, single-chip radio transceivers can have transmission speeds of 1Mbps (which is the case for the nRF2401 used in the IMEC platform). Considering all the above it is possible to see the dramatic reduction in transmission time that ShockBurst mode can achieve as the data would be now transmitted in one burst and at 1Mbps, resulting in a transmission time of 1ms. Figure 2.6 shows this example, and figure 2.7 shows the difference between the transmission time of ShockBurst mode and a traditional device wihout this technology. The reduction in power consumption comes from the fact that the Radio stays in the costly transmit mode for less time as it doesn't idle-wait for data to arrive at slow rates restricted by the MCU. In addition to the reduced energy consumption, shorter transmission times also help to reduce the risk of on air collisions.

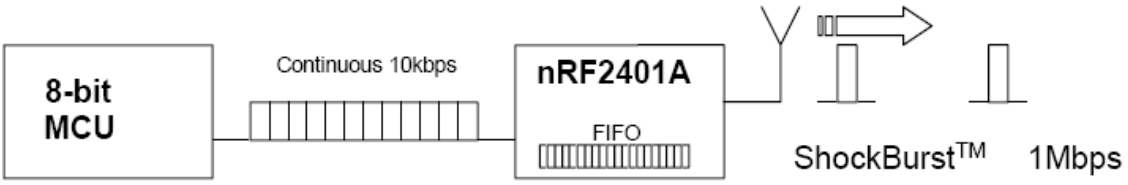

Figure 2.6. Clocking in data with MCU and sending with ShockBurst technology

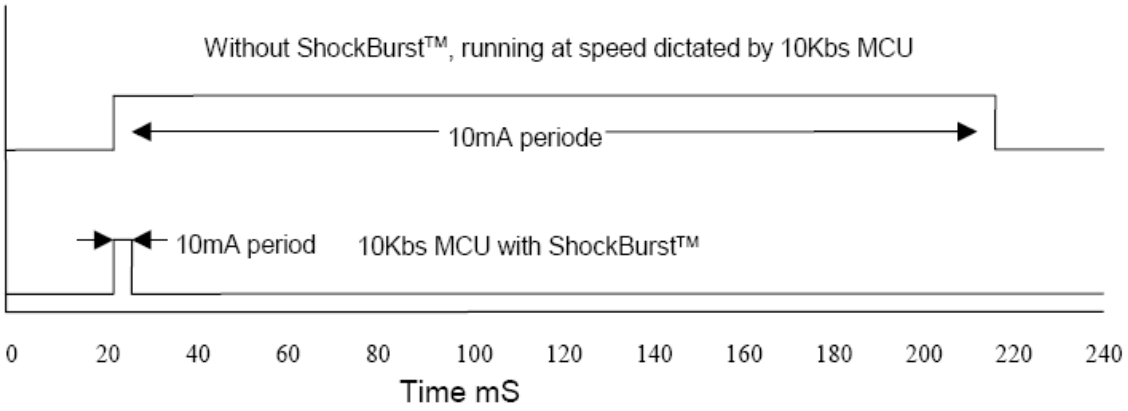

Figure 2.7. Current consumption with & without ShockBurst technology

The ShockBurst approach to the radio communications is very attractive, as it reduces overheads at the MCU when communicating data and also saves energy. There are though some limitations which result from its design that might make it inappropriate for use in certain situations:

- There is no obvious way of implementing Broadcasts or a Promiscuous mode since every receiver has its own address and the chip discards everything but packets addressed to it.
- There is a limit of 256 bits maximum ShockBurst frame length (including Address and CRC bits) which may require fragmentation and reassembly to be performed in upper layers of the protocol stack.

- There is absolutely no way of determining whether a packet was received but corrupted (CRC check failed) or was not received at all.
- There is no way of performing Carrier Sense since the chip will not provide any information about ongoing transmissions unless what is on air is addressed to it.

In our model we only consider ShockBurst mode because it is the only mode included in the real platform used as case study, because there is no need for communication in the direction from the base station to the nodes. Thus, this implementation using the low frequency crystal included in the design achieves the theoretically best low power consumption.

2.4 Other hardware platforms

Apart from the platform that we have chosen to validate our model, there are some other platforms that provide similar functionality although they are in general bigger in size. The most important are Mica and Telos motes, both developed by University of California, Berkeley and distributed by Crossbow, which will be described below.

2.4.1 Mica motes

Mica was the platform that TinyOS was originally built around and as expected, it has significantly influenced the OS in its initial releases (TinyOS was initially running only on Micas). There are some versions of this platform (Mica, Mica2, Mica2Dot, etc.).

The Mica2 motes use the Chipcon CC1000, FSK modulated radio. All models utilize a powerful Atmega128L microcontroller and a frequency tunable radio with extended range. Figure 2.8 shows the Mica2 mote.

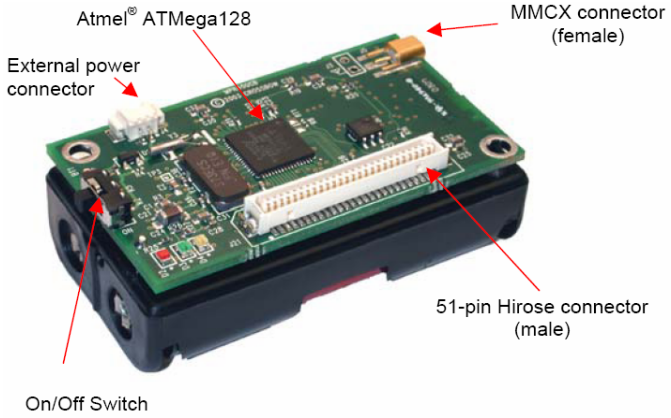

Figure 2.8. Photo of a Mica2 without an antenna.

The MicaZ is the latest generation of Motes from Crossbow Technology. The MPR2400 (2400 MHz to 2483.5 MHz band) uses the Chipcon CC2420, IEEE 802.15.4 compliant, ZigBee ready radio frequency transceiver integrated with an Atmega128L microcontroller. The same Mica2, 51 pin I/O connector, and serial flash memory is used; all Mica2 application software and sensor boards are compatible with the MPR2400. This platform is depicted in figure 2.9.

Figure 2.9. Photo of the MPR2400—MicaZ with standard antenna.

2.4.2 Telos motes

Telos motes use a MSP430 microcontroller, from Texas Instruments, the same microcontroller that IMEC designers chose for the Sensor Cubes. Telos uses the Chipcon CC2420 radio, the same radio as MicaZ, that is IEEE 802.15.4 compliant. This new design is based on experiences with previous motes generations and tries to achieve three major goals to enable experimentation: minimal power consumption, easy to use, and increased software and hardware robustness. Figure 2.10 shows a Telos mote.

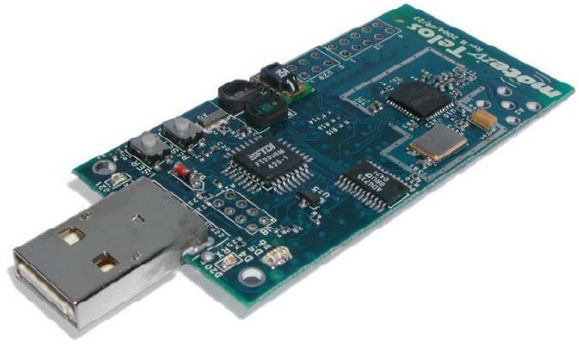

Figure 2.10. Telos wireless module with IEEE 802.15.4 wireless transceiver.

2.5 Rationale for choosing Sensor Cubes

We chose the Sensor Cubes for our project for the following reasons:

- The ultra-low power MSP430 microcontroller from Texas Instruments. This chip is one of the most low-power consuming microcontrollers now available. It is able to work at 8MHz maintaining a very low energy consumption. It is also equipped with an on-chip 12-bit A/D converter that speeds up the communication between microcontroller and sensors.
- The Nordic nRF2401 Radio Transceiver, that also contributes to the low energy consumption of the whole device. This is mainly due to the ShockBurst mode of operation, that allows the radio to be switched on for a short period of time when it wants to transmit a packet. ShockBurst mode of operation presents some deficiencies, like it is impossible to implement a “carrier sense” MAC protocol,

but we can always use the Direct mode of operation and the behavior of the radio chip will be the same as a traditional radio transceiver, and we will be able to implement this type of protocols.

- Its architecture, which divides the node in different layers. This architecture allows us to change, include or remove some of these layers in order to configure our own device. The sensor layer included in the model that we have has humidity, temperature and luminosity sensors, but depending on the application we can change this sensor layer for another one thas is equipped with other sensors.

- Cooperation between different centres. There are currently three important organizations involved in this project: Holst, EPFL and UCM. Holst is a centre that was set up by IMEC and TNO, located in Eindhoven (Holland). Holst is responsible of the Sensor Cubes design and provides us the hardware that we need to carry out our research. EPFL (Ecole Polytechnique Fédérale de Lausanne) is a Swiss university, there are some students in this centre that are in charge of researching about energy harvesting and porting the new version of TinyOS to the nodes. At UCM (Universidad Complutense de Madrid) there is also a group of people developing drivers for the new sensor modules that will be included in the new releases of the nodes and applications for ECG monitoring. EPFL and Holst help us to do something that we can not do at UCM: design and build a sensor node, modify it and rebuild it again. This collaboration among different organizations allows us to increase the scope of the work carried out.

Chapter 3
Software Platform

The Sensor Cubes were supplied with some basic low-level C libraries. Using these libraries, the nodes can be programmed and it was possible to establish a communication between a Sensor Cube and the base station.

An operating system was included in the nodes, to provide to programmers an easy way to implement applications and cover the lacks of the C libraries previously used (scheduling tasks, performing memory management, etc.). With an operating system, they don't have to worry about the platform that the application is going to be executed in or about other hardware dependent details. The operating system used in the nodes is called TinyOS [11] (we use the version 1.1.13), it also includes a simulator called TOSSIM [12] which we can use to test and debug applications in a computer before loading them in the nodes. This operating system and its simulator are described in the following subsections, we also give an overview of other operating systems for WSN.

3.1 TinyOS

Traditional embedded operating systems are typically large (requiring hundreds of KB or more of memory), general-purpose systems consisting of a binary kernel with a rich set of programming interfaces. Examples include WinCE [25], QNX [26], PalmOS [27], pSOSystem [28], Neutrino [29], OS-9 [30], LynxOS [31], Symbian [32], and uClinux [33]. Such operating systems target devices with greater CPU and memory resources than sensor network nodes, and generally support features such as full multitasking,

memory protection, TCP/IP networking, and POSIX-standard APIs that are undesirable (both in terms of overhead and generality) for sensor network nodes.

TinyOS is perhaps the first operating system specifically designed for wireless sensor networks. It is developed by a consortium led by University of California, Berkeley in co-operation with Intel research. TinyOS is based on an event based operating environment designed for using with embedded networked sensors. When an external event occurs, such as an incoming data packet or a sensor reading, TinyOS calls the appropriate event handler to handle the event.

This operating system has been written using the nesC programming language [13], which is an extension to the C language optimized for the memory restrictions of sensor networks. In nesC, *components* are the basic building blocks of an application and contain code which provides the functionality described in corresponding *interfaces*. Creating an application in TinyOS mainly involves the coding of the modules that will contain the application code and creation of a *configuration* that dictates which components are used by the application and how they will interact, this approach actually goes down to the operating system itself and, depending on the application it is compiled with, it only uses the required modules every time. A good description of the TinyOS as a whole is "a set of reusable system components along with a task scheduler".

As well as presenting some differences with the C programming language, nesC has some restrictions in order to do the code more robust and efficient. The main differences are:

- NesC does not allow the use of function pointers, thus making the call graph of a program known at compile time. This allows optimization by reducing message passthrough between components.
- Dynamic memory allocation is not supported and the components of a program can only statically declare, preventing thus memory fragmentation and errors due to allocation failures at runtime.

TinyOS is designed to support the concurrency intensive operations required by networked sensors with minimal hardware requirements.

TinyOS 2.0 [34] is already available and it will be ported to the Sensor Cubes early. This new version is a clean slate redesign and re-implementation of TinyOS. Its development was motivated by the belief of developers that many aspects of 1.x strain to meet requirements and uses that were not foreseen when it was designed and implemented. The structure and interfaces 1.x defines have several fundamental limitations. While these limitations can be worked around, this practice has led to tightly coupled components, hard to find interactions, and a very steep learning curve for a newcomer to sensor network programming. Some of the new version of TinyOS improvements are:

- Platform/hardware abstraction: the organization of the platforms in TinyOS 2.0 is done in a more modular way.
- Scheduler: In TinyOS 1.x, there is a shared task queue for all tasks, and a component can post a task multiple times. In TinyOS 2.0, every task has its own reserved slot in the task queue, and a task can only be posted once. Applications can also replace the scheduler, if they wish, allowing programmers to try new scheduling policies.
- Timers: TinyOS 2.0 provides a much richer set of timer interfaces than 1.x.
- Communication: in TinyOS 2.0, there is a new type of message buffer that is large enough to hold a packet from any of a node's communication interfaces.
- Network protocols: TinyOS 2.0 provides simple reference implementations of two of the most basic protocols used in mote networks: dissemination and collection. Dissemination reliably delivers small (fewer than 20 byte) data items to every node in a network, while collection builds a routing tree rooted at a sink node. Together, these two protocols enable a wide range of data collection applications. Collection has advanced significantly since the most recent beta release; experimental tests in multiple network conditions have seen very high (>98%) deliver rates as long as the network is not saturated.

3.2 Other operating systems for WSN

3.2.1 Contiki OS

Contiki [35] is a small, open source, highly portable, multitasking computer operating system developed for use on a number of memory-constrained networked systems ranging from 8-bit computers to embedded systems on microcontrollers, including sensor network motes.

Contiki is designed for embedded systems with small amounts of memory. It consists of an event-driven kernel on top of which application programs are dynamically loaded and unloaded at runtime. This operating system processes use light-weight protothreads that provide a linear, thread-like programming style on top of the event-driven kernel. It also supports per-process optional preemptive multi-threading, interprocess communication using message passing through events, as well as an optional GUI subsystem with either direct graphic support for locally connected terminals or networked virtual display with VNC or over Telnet.

Contiki runs on a variety of platform ranging from embedded microcontrollers such as the MSP430 and the AVR to old home PCs. Code footprint is on the order of kilobytes and memory usage can be configured to be as low as tens of bytes.

3.2.2 SOS

SOS [36] is an operating system for mote-class wireless sensor networks developed by the Networked and Embedded Systems Lab at UCLA. SOS uses a common kernel that implements messaging, dynamic memory, module loading and unloading, and other services. SOS uses dynamically loaded software modules to create a system supporting dynamic addition, modification, and removal of network services.

Dynamic reconfigurability is one primary motivation and goal for SOS. In the domain of wireless sensor networking, reconfigurability is the ability to modify the software on individual nodes of a sensor network after the network has been deployed and initialized. This provides the ability to incrementally update the sensor network after it is deployed,

add new software modules to a sensor node after deployment, and remove unused software modules when they are no longer needed.

Creating an expressive system that provides programmers with commonly needed services is a second primary goal for SOS. Many sensor network applications need access to the same primitives ranging from memory pools for easy data packet management to application diagnostic network services such as neighborhood discovery.

To these ends, SOS takes traditional operating system ideas and transforms them into a form appropriate for the resource-constrained domain of wireless sensor nodes. This results in a flexible system that can be used to write clean and easy to maintain sensor network node systems.

3.3 TOSSIM

TOSSIM [12] is a discrete event simulator for TinyOS sensor networks and is part of our energy simulation tool. Instead of compiling a TinyOS application for a node, users can compile it into the TOSSIM framework, which runs on a PC. This allows users to debug, test, and analyze algorithms in a controlled and repeatable environment.

When we develop an application for TinyOS, we can compile that application for any of the platforms installed in our TinyOS distribution. All the installed platforms can be found in $TOSROOT/tos/platforms, there is a directory for each one. For instance, “mica” or “telos” are the platform for Mica and Telos motes, respectively. If we want to compile our application for the Mica motes, we have to execute the command “make mica”. There exists also a platform for TOSSIM, that is called “pc”. An application is compiled for the simulator by executing “make pc”, this generates an executable file that can be launched, specifying the number of nodes that will take part in the simulation and some other parameters like simulation time, a different radio model, etc.

As TOSSIM runs on a PC, users can examine their TinyOS code using debuggers and other development tools. TOSSIM can simulate thousands of nodes simultaneously. Every mote in a simulation runs the same TinyOS program.

This is possible by replacing the lower level modules of TinyOS that communicate directly with hardware, with modules that emulate the behavior of this hardware on the PC. The emulated hardware in TOSSIM includes the following modules:

- Analog-to-Digital Converter (ADC)
- Clock
- EEPROM
- Boot sequence component
- Radio Stack components
- Sensing modules

The last version of the simulator is called PowerTOSSIM (it is included in TinyOS version 1.1.9 and beyond). The main difference regarding the previous version of TOSSIM is the inclusion of the assessment of the power consumed by the application. PowerTOSSIM includes a model of the power consumption of the Mica2 nodes [37]. The Mica2 platform is composed of:

- 7.3 MHz ATmega128L processor
- 128KB of code memory, 512KB EEPROM
- 4KB of data memory
- ChipCon CC1000 radio capable of transmitting at 38.4 Kbps with an outdoor transmission range of approximately 300m.

However, this model is not accurate enough, it does not take into account some important parameters that affect to the energy consumption. These parameters will be analyzed in depth in the following section, where we propose our optimized energy model which covers the lack of accuracy of the model developed for the Mica motes.

Before including our model, PowerTOSSIM was modified to perform online power management. With this modification, energy assessment is accelerated by doing it online instead of doing it after running the whole simulation.

When we compile an application we have to choose the platform which the executable file generated is going to run in. All the platforms which we can compile our applications for, are contained in the installation of TinyOS. We have modified the simulator to make it able to imitate the behavior of our platform. We add more detail to the IMEC model to simulate the energy factors explained in next section. A new model has been added to the platform "pc", that is the one used to compile the applications for TOSSIM. This model imitates the behavior of our node. With this modification we can user PowerTOSSIM to simulate our nodes. The model that we propose is much more accurate that the one initially included in PowerTOSSIM.

3.4 Other simulators for WSN

3.4.1 ns-2

Ns-2 [38] is a discrete event network simulator. It is popular in academia for its extensibility (due to its open source model) and plentiful online documentation. This is very popular for simulation of routing and multicast protocols, and it is heavily used in ad-hoc research. Ns-2 supports a set of well known network protocols, offering simulation results for wired and wireless networks alike.

This simulation tool was built in C++ and provides a simulation interface through OTcl, an object-oriented dialect of Tcl [39]. The user describes a network topology by writing OTcl scripts, and then the main ns-2 program simulates that topology with specified parameters.

Ns-2 does not scale well for sensor networks, due to its object-oriented design. Every node is its own object and can interact with every other node in the simulation, creating a large number of dependencies to be checked at every simulation interval, leading to an n^2 relationship. Another drawback to ns-2 is the lack of customization available: packet formats, energy models, MAC protocols, and the sensing hardware models all differ from those found in most sensors. Other disadvantage is that it does not model application behavior, in many network environments this is not a problem, but sensor networks often contain interactions between the application level and the network protocol level.

3.4.2 OPNET

OPNET [40] is a discrete event, object-oriented, general purpose network simulator. It uses a hierarchical model to define each aspect of the system. The top level consists of the network model, where topology is designed. The next level is the node level, where data flow models are defined. A third level is the process editor, which handles control flow models. Finally, a parameter editor is included to support the three higher levels.

The results of the hierarchical models are an event queue for the discrete event simulation engine and a set of entities representing the nodes that will be handling the events. Each entity in the system consists of a finite state machine which processes the events during simulation. The simulation tool is capable of recording a large set of user defined results.

Unlike ns-2, OPNET supports the use of modeling different sensor-specific hardware, such as physical-link transceivers and antennas. It can also be used to define custom packet formats. The simulator aids users in developing the various models through a graphical interface. The interface can also be used to model, graph, and animate the resulting output.

OPNET suffers from the same object-oriented scalability problems as ns-2. It is not as popular as ns-2, at least in research being made publicly available, and thus does not have the high number of protocols available to those simulators. Additionally, it is only available in commercial form.

Chapter 4
Energy model

The nodes that constitute a WSN get energy from a battery, and possibly from an energy scavenger, then, the life of a node is limited by the life of its power supply. In some cases, the nodes are spread in hostile or non easily accessible areas and it is very difficult (or even impossible) to replace the battery. In healthcare applications happens the same, if a patient has some nodes around his body, it can be not very comfortable for him to change the batteries of the nodes. In the future, perhaps these nodes will be implanted in the body, doing the task of changing the batteries even more unpleasant for the patient. For these reasons, battery life, and therefore, energy consumption is a high priority issue in WSN.

Analyzing the node energy consumption we have measured that the part of the nodes which consumes more energy is the radio chip (typically between 70% and 90% of the total energy consumed by the node, depending on the application). With these results we conclude that a good energy model should be centered in the energy consumed by the radio, which is not carried out by any simulation tool yet. We have to try to model the radio as well as possible and take into account all the parameters that can influence the final value of energy consumed. The following sources are major causes of energy waste in the radio:

- *Collisions*: when two packets are transmitted at the same time and collide, they become corrupted and must be discarded and transmitted again. But, PowerTOSSIM models collisions as a logical or, there is no cancellation. This

means that distance does not affect signal strength; if mote B is very close to mote A, it cannot cut through the signal from far-away mote C.

- *Idle listening*: it happens when the radio is listening to the channel to receive possible data. The waste of energy due to this factor can be very high, especially in applications where there is no data to send during the period when nothing is sensed.
- *Overhearing*: this is produced when a node receives packets that are destined to other nodes. The node that is overhearing consumes power receiving the packet and discarding it.
- *Control packet overhead*: sending, receiving and listening for control packets consume energy. Since control packets do not carry data, they reduce the throughput.

Current models for Wireless Sensor Networks do not take into account all these factors that affect to the energy consumption very significantly as we demonstrate in the case study. Then we have added to our model all these factors. Now, we are going to analyze separately the four factor previously described. For each factor, firstly we will see the deficiencies of the previous model included in TOSSIM for the Mica motes and after that we will present our solution to the problem.

- Collisions are modeled by TOSSIM as a logical or, as we said before, then a node can receive corrupt packets. In the previous model, it was assumed that all the packets reach the destination without errors, then it was absolutely impossible to detect whether a collision had happened. In our case, collisions can be detected by the simulator in the same way as they are detected in the real hardware. When a node wants to send a packet, the microcontroller of the node sends the payload to the Nordic nRF2401, the radio chip adds a preamble, the destination address and CRC. When the packet is received by a node, it checks the packet integrity using the CRC added by the sending node and it discards the packet if there is any error. This behavior of the radio chip is fully covered by our model. With this characteristic we can detect collisions and discard the

packets involved in it, while we take into account the energy consumed by the reception of the corrupted packets.

- The energy consumed for idle listening is taken into account by TOSSIM, we did not improve the simulator in this aspect, because it is accurate enough regarding this factor.
- Overhearing happens in our architecture, but in a different way. In the case of traditional radio transceivers, when a packet is received it is sent to the microcontroller, which is the component in charge of checking the address, CRC, etc. The Nordic nRF2401 checks the address and CRC before sending the packet to the microcontroller, as we explained before for the collisions. This special behavior is also added to our model. When a node receives a packet, it checks the destination address included by the sending node. If both the packet address and the receiving node address are the same, the packet is given to the microcontroller. On the contrary, if both addresses are different, the packet is discarded and the microcontroller will not notice the reception of that packet.
- The last important factor in the energy consumption, control packet overhearing, is not covered by current models. In fact, the original model contained in TOSSIM for the Mica motes do not even consider acknowledgements (ACKs), which are very influent in the power consumed by a network for the following reasons:
    - Deep down, an acknowledgement is a packet, then if we do not take into account ACK transmissions, we are not considering a lot of packet transmissions, that have a very significant impact in the value of the total energy consumed.
    - ACKs can also collide with other packets. Usually, when this happens, the packet that the lost ACK confirms is retransmitted, and consequently the ACK is also retransmitted, in order to try to confirm the new transmission of the packet. This also has a big influence in the final value of energy consumed by the node.

In the model that we propose, control packets are also included, and then all the consequences that these packet transmissions can have in the total value of energy consumed by the device, like collisions, overhearing, etc.

Table 4.1. Power model for the Sensor Cube. The node was measured with a 2.4V power supply.

Mode	Current	Mode	Current
CPU		**Radio**	
Active	1.0 mA	Rx	18.0 mA
Idle	1.0 mA	Tx (-20 dBm)	8.8 mA
ADC Noise Reduce	1.0 mA	Tx (-19 dBm)	9.0 mA
Power-down	0.001 mA	Tx (-15 dBm)	10.0 mA
Power-save	0.001 mA	Tx (-8 dBm)	11.0 mA
Standby	0.001 mA	Tx (-5 dBm)	12.0 mA
Extended Standby	0.001 mA	Tx (0 dBm)	14.0 mA
LEDs	2.5 mA	Tx (+4 dBm)	14.0 mA
Sensor board	0.001 mA	Tx (+6 dBm)	14.0 mA
EEPROM access		Tx (+8 dBm)	14.0 mA
Read	3.0 mA	Tx (+10 dBm)	14.0 mA
Write	3.0 mA		

The microcontroller consumes a very little amount of energy, in fact, in all the models the microcontroller is discarded. In ours, the microcontroller is not simulated at the bit level, because it would increase too much the time needed to run a simulation, but we do a very good approximation that results of multiplying the number of cycles that the simulation takes by the average power consumed by the microcontroller in a cycle. This can be done because of the differences among radio and microcontroller energy consumption.

Table 4.1 presents the resulting power model for the Sensor Cube hardware platform, these values have been measured directly in the node, with a 2.4V power supply. As the

table shows, the different CPU power modes cover a wide range of current levels, from 0.001mA in the “power down” state up to 1.0mA when actively executing instructions. Likewise, the choice of radio transmission power affects current consumption considerably, from 8.8mA at -20dBm to 14.0mA at +10dBm.

Chapter 5
Case study

Our case study is centered on Wireless Sensor Networks for medical applications. One interesting application, in this respect, is the real-time remote and accurate analysis of human heart activity, which has always been a challenging problem for biomedical engineers. Heart disorders like Cardiovascular Disease (CVD) and stroke remain by far the leading cause of death in the world for both women and men of all ethnic backgrounds [41].

Our nodes are equipped with sensors that are able to perform EEG/ECG (electrocardiogram/electroencephalogram) which can readily reveal a number of heart malfunctions.

This kind of applications usually consists in a small number of nodes (between 5 and 10, depending on the application) with a star topology, as it is shown in figure 5.1. The nodes are heterogeneous, some nodes can perform ECG while other are performing EEG or other tests. Radio transmissions are very short range because of the proximity between nodes. In order to perform a reliable ECG or EEG test we need to sample at 1KHz, for this reason the throughput of these networks is very high, we send 1 measurement/ms to the base station in the case of ECG and 24 measurements/ms in the case of EEG, because EEG samples 24 channels. Figure 5.2 shows a typical ECG application that uses 3 leads.

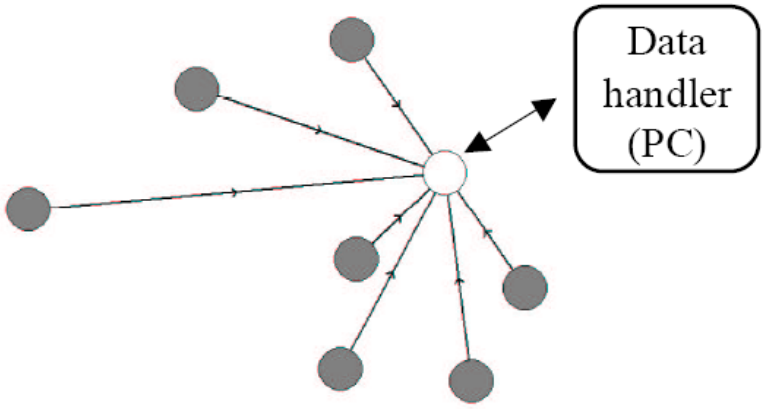

Figure 5.1. Star topology sensor network.

Figure 5.2. Wireless 3-lead ECG sensor

TinyOS was ported to the Sensor Cubes using the ShockBurst mode. The MAC protocol implemented is a very simple ALOHA with acknowledgements [42]. With this protocol, when a node has to send a packet it does not sample the medium to know if other node is transmitting at the same time, the node simply sends the packet. For this reason, the probability of receiving corrupt packets with ALOHA is very high due to the collisions that are very probable using this protocol, especially in the applications that we are using. Due to the high throughput of the network is very easy to have collisions between packets, that will be heard and discarded, increasing the energy consumption.

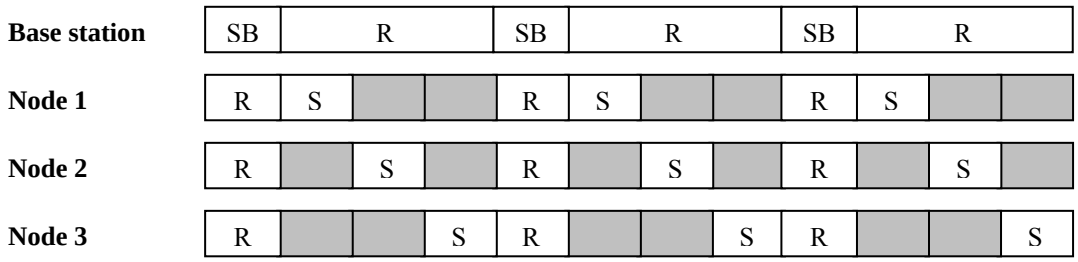

Figure 5.3. Scheme of operation of a TDMA protocol.
Shady= the radio is switched off. SB = Sending Beacon, RB = Receiving Beacon, R = Receiving, S = Sending data

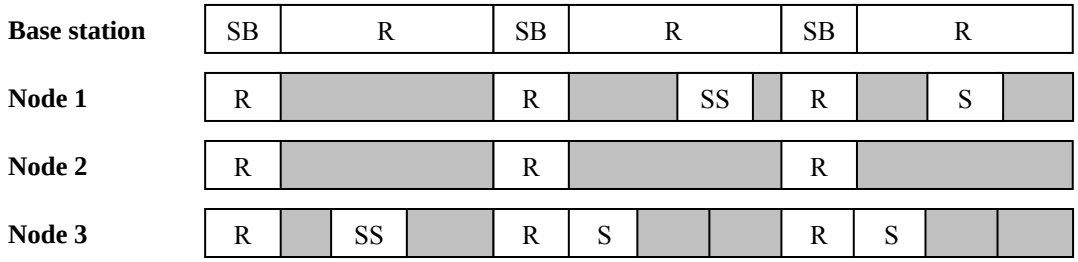

Figure 5.4. Scheme of operation of the first implementation of our TDMA protocol.
Shady= the radio is switched off. SB = Sending Beacon, RB = Receiving Beacon, R = Receiving, S = Sending data, SSR = Sending Slot Request

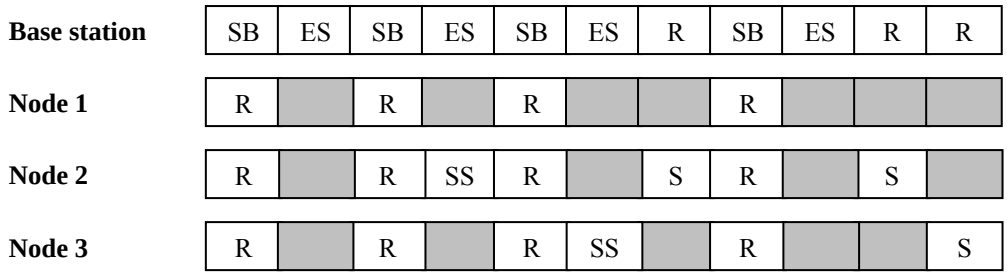

Figure 5.5. Scheme of operation of the second implementation of our TDMA protocol.
Shady= the radio is switched off. SB = Sending Beacon, RB = Receiving Beacon, R = Receiving, S = Sending data, SSR = Sending Slot Request, ES = Empty Slot

In the case study we use our energy model to compare different MAC protocols in terms of energy consumption. We have implemented a TDMA protocol to reduce energy

consumption and we improved this protocol making it more dynamic. The simulations results of these two TDMA protocols are presented in this section.

TDMA (Time Division Multiple Access) is a MAC protocol which allows several users to share the same frequency channel by dividing the signal into different timeslots. The users transmit in rapid succession, one after the other, each using his own timeslot. This allows multiple stations to share the same transmission medium (e.g. radio frequency channel) while using only the part of its bandwidth they require. Figure 5.3 represents the TDMA scheme for 3 nodes and a base station. Rectangles in grey mean that the radio is in standby mode, SB = Sending Beacon, RB = Receiving Beacon, R = Receiving, S = Sending data. As we can see, the base station sends beacons regularly to signal the beginning of a TDMA cycle to the nodes, the rest of the time is listening to possible transmissions from the nodes. Each node has a slot assigned, if the node has data to send, it sends the data in its slot.

The first TDMA protocol implemented to test the reliability of our model has the same behavior as the one described in the previous paragraph. At the beginning of the simulation, all the slots are free and every node has to send a slot request to the base station, the base station informs the nodes that have ask for a slot in the following beacon. Each node sends the request slot when it is turned on, and resends the request after some time if the base station does not assign it a slot. Once a slot is assigned to the node, it transmits data to the base station in that slot and receives the acknowledgements of those transmissions in the beacons. If the acknowledgement corresponding to a packet transmission is not received after some time, the node resends the packet. In figure 5.4 we can see how the slots are assigned (SSR = Sending Slot Request). In the first TDMA cycle, the node 3 sends a slot request, the following beacon informs it that the first slot has been assigned to it. In the second TDMA cycle, the same thing happens with node 1, which is informed in the following beacon about the slot assigned to it.

We modified this protocol making it more dynamic. The nodes have a short life time, compared with regular computers, then a more dynamic protocol is needed in order to not waste bandwidth, and therefore energy, when a node is dead. It can be used also to increase the number of nodes of the network, in the protocol described previously, we

have to know in advance the maximum number of nodes to calculate the length of the TDMA cycle and the slots. The protocol that we are going to describe now adapts itself dynamically to the number of nodes of the network. At the beginning, the base station sends a beacon and leaves an empty slot after the beacon transmission. This empty slot is used only for slot requests, when a node wants to request a slot, it sends the request in the empty slot. In the following beacon, the node is informed about the slot assigned to it. The empty slot is always the one after the beacon, the slots dedicated for transmission are located after the empty slot. Figure 5.5 depicts how slots are assigned in the new implementation (ES = Empty Slot). In the second TDMA cycle, the node 2 sends a slot request, the following beacon informs it that the first slot has been assigned to it. The third TDMA cycle is longer, because there is a slot after the empty one dedicated to the node 2, in this TDMA cycle, the node 3 sends a slot request (also in the empty slot) and it is informed in the following beacon of its slot assignation. We see again that the length of the TDMA cycle has increased due to the assignation of another slot for the node 3. In this implementation is relatively easy that two nodes request a slot at the same time, if this happens, the packet received by the base station in the empty slot will be a corrupted packet and it will have to be discarded. We try to avoid this problem initializing randomly a variable that indicates the number of cycles that the node has to wait before requesting the slot, this variable is decremented when a beacon is received and when its value is 0, the node requests a slot in the following empty slot. With this implementation is easier to establish the topology, as we are going to see in the experimental results, because slots are assigned to the nodes faster than in the first implementation.

5.1 Experimental results

Table 5.1 contains values obtained with PowerTOSSIM using the model for the Mica2 motes and the TDMA protocol with the model for the Sensor Cubes. We have run simulations of 60 seconds of execution time. The network simulated consists in 6 elements (5 nodes and a base station). We chose this configuration of the network because is used in ECG applications, as benchmark we use an application called SurgeImec. In SurgeImec, each node takes sensor readings and forwards them to a base

station. For the first implementation we choose a TDMA cycle of 6 slots (one for the beacon and five for the nodes).

Table 5.1. Energy consumption for the initial TOSSIM energy model for Mica2 motes and the 2 different versions of the TDMA protocol with the new model. The application was executed for 60 seconds and all values are in milijoules (mJ).

	Base station	**Nodes (average)**
Initial TOSSIM model	2007.57	2077.23
First TDMA	2728.48	1121.84
Second TDMA	2432.36	700.52

Table 5.2. Simulated component energy breakdown. All values are in milijoules (mJ).

	CPU	**Radio**	**ADC**	**Leds**	**Total**
Base station (initial TOSSIM model)	742.86	1264.70	0.00	0.00	2007.57
Nodes (initial TOSSIM model) - average	714.89	1180.13	0.00	182.21	2077.23
Base station (first implementation)	143.60	2584.88	0.00	0.00	2728.48
Nodes (first implementation) - average	137.58	732.16	87.71	164.37	1121.84
Base station (second implementation)	143.91	2288.44	0.00	0.00	2432.36
Nodes (second implementation) - average	138.11	427.08	85.82	54.92	700.52

In the results showed in table 5.1 we can appreciate that the energy consumed by the base station with the first implementation is slightly higher than with the second one, this is due to that the base station in the first implementation is one slot in TX mode and 5 slots in RX mode, while the second implementation adapts the TDMA cycle length to the nodes that are ON and, especially at the beginning, the energy consumption is lower because the proportion of time that the radio is listening (the mode in which the radio

consumes more energy) is lower. If we look at the values of energy consumed by the nodes we can see that they are also lower in the second implementation, which is basically due to the reduction in the number of collisions when different nodes request a slot.

In table 5.2 we have the energy values of the previous simulations more detailed. We specify the energy consumed by each component of the node. We observe in this table that the energy consumed by the CPU is almost the same in all the cases. With the radio does not happened the same, in both implementation we appreciate a big difference between the base station and the nodes. The node's radio consumed less energy because it is ON only when it sends a packet or when it is waiting for listening to the beacon, the rest of the time the radio is OFF.

Regarding the differences with the Mica2 motes model, we can extract the following conclusions from the results shown in both tables:

- With both implementations of the TDMA protocol, we can appreciate a big difference between the values of energy consumed by the nodes and the base station. The base station is all the time ON, while the nodes are ON only when they receive the beacon and when they want to send information. This does not happen with the Mica2 model, because there is no MAC protocol included, therefore the base station and the nodes consume approximately the same energy.
- The Mica2 model does not take into account the energy consumed by the ADC, this is another deficiency of the model, because this component has influence on the final value of energy consumed.
- The base station consumes more energy with the new model. This is due to control packets, which are not considered by the Mica2 energy model.

5.2 Modular implementation of the driver

As we previously referred, the MAC protocol originally included in the TinyOS porting for the Sensor Cubes was a very simple ALOHA. When we implemented the less dynamic TDMA for TOSSIM we came up with serious difficulties to replace the

ALOHA with the new protocol. A lot of difficulties appeared again when we ported the TDMA to the Sensor Cubes, because of the lack of modularity in the structure of the driver. Some files need to be modified when we implement a new MAC protocol and when we want to port an implementation of a MAC protocol from TOSSIM to the Sensor Cubes or vice versa. To get rid of this kind of problems, we propose a new architecture for the nRF2401 radio driver, which is depicted in figure 5.6.

The green blocks are configurations, used by TinyOS to link different components, which are represented by white blocks in the structure.

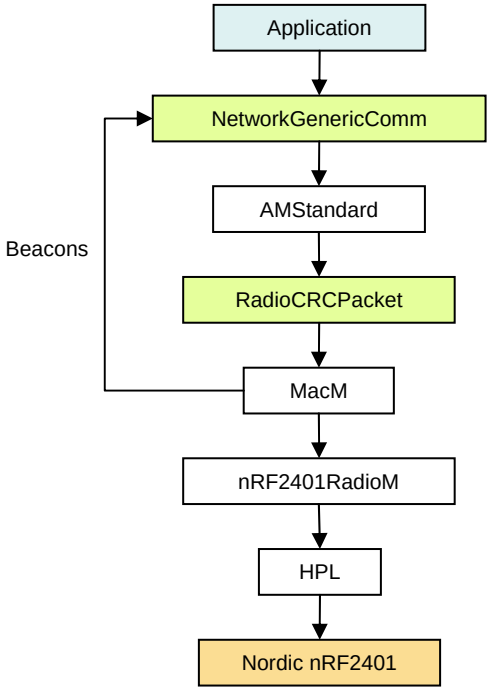

Figure 5.6. Structure overview of the nRF2401 driver for TinyOS 1.0.

The NetworkGeneriComm configuration defines the components used for both USART and radio communication, and links both with the AMStandard component.

The AMStandard component encapsulates radio packets in the AM format. Radio communication in TinyOS follows the Active Message (AM) model, in which each packet on the network specifies a handler ID that will be invoked on recipient nodes.

Think of the handler ID as an integer or "port number" that is carried in the header of the message. When a message is received, the receive event associated with that handler ID is signaled. Different motes can associate different receive events with the same handler ID [43].

RadioCRCPacket configuration is the bridge between Network and MAC layer, it can also be used to skip the MAC layer and directly access the HAL (see comments in RadioCRCPacket.nc for more details).

The MacM component defines the MAC layer and communicates to the lower layer with 4 interfaces, StdControl, BareSendMsg, ReceiveMsg and MacStdControl. The first three are default TinyOS interfaces while the third has been created in order to control the radio power modes from the MAC layer (these functionalities are not implemented yet in the MAC protocol but they will be added in the early future). The MAC layer exchanges AM packets with the HAL through BareSendMsg and ReceiveMsg interfaces. The MacM component is also connected back to the AMStandard component because of the beacons, also sent in AM format.

The nRF2401RadioM component defines the HAL of the radio. It provides the upper level layer (MAC) with initialization, send and receive functions. The HAL receives AM packets from the MAC layer, converts them in bytes and sends them to the HPL. The same operation is performed with received packets. All the functions to control the radio power states are not performed in this layer; therefore the calls from the MAC layer (through the MacStdControl interface) are just passed to the HPL level.

The HPL layer (which includes the HPLnRF2401M and nRF2401Control components) links the hardware component with the HAL and performs all the operations needed to control the radio device at register level. This layer has been developed at UCL and seems to work properly. If needed, some further optimizations will be performed.

Figure 5.7 shows the connections between the 3 lower level components of Figure 5.6. The white boxes represent the modules and the green ones the interfaces used. All the connections are defined in the configurations files MacC.nc and nRF2401RadioC.nc.

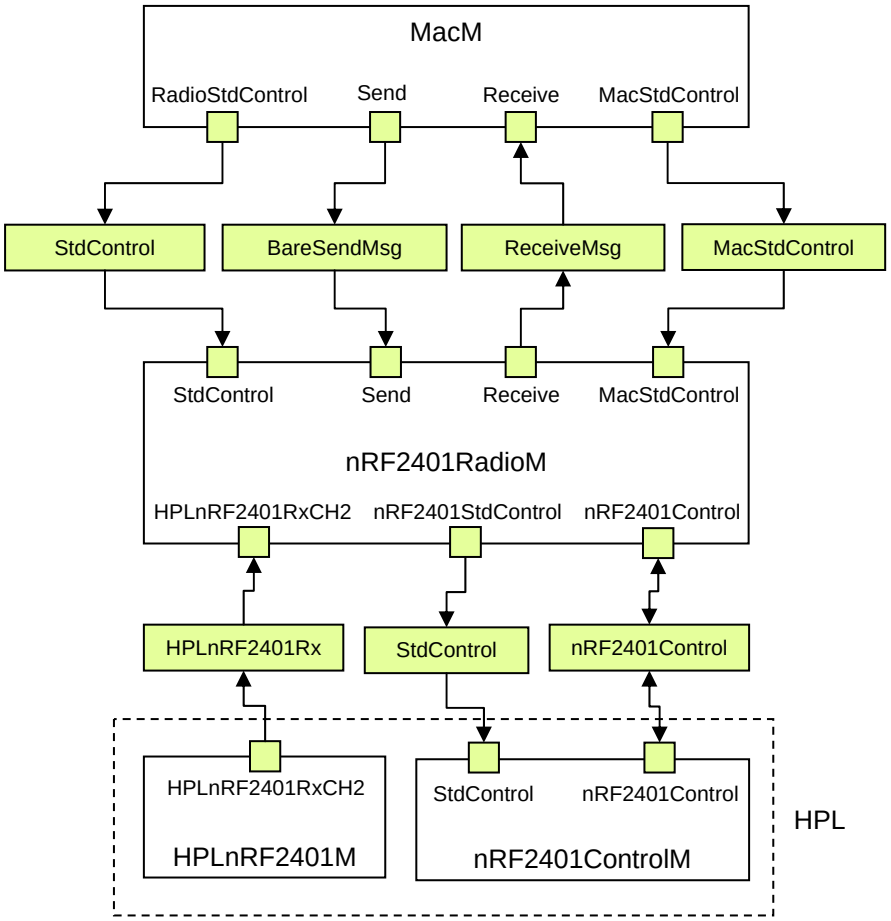

Figure 5.7. Interfaces between the low level components of the driver

Chapter 6
Future work

This project began one year ago, therefore it is in a very preliminary stage. Some of the initial objectives were fulfilled:

- The physical layer of the radio is modeled in the simulator.
- We have an accurate model for the Sensor Cubes to make energy consumption estimations.

But there are other goals for the future:

- A couple of applications for ECG monitoring are being ported to the ECG/EEG version of the Sensor Cubes and will be available very soon. For the next course, two more complex algorithms for ECG signal processing will be also ported to the nodes.
- The task of porting TinyOS 2.0 to the Sensor Cubes is being studied and it will be tackled in the following months.
- The rest of layers of the architecture will be also modeled, this will allow us to propose changes in the current architecture to adapt it to the requirements of concrete applications.

Chapter 7
Conclusions

Nowadays, WSNs are growing in size, complexity and fields of application. Therefore, it is becoming more difficult to debug applications, test new improvements in the nodes, etc. The experience has proved that flexible WSN simulators are essential to achieve these goals and make easier testing different final configurations of WSNs (e.g. topologies, communication protocols, etc.). In addition, reliable and accurate node models for these simulators are required to accurate simulate the behavior of our target architecture without waiting for the final implementation of the nodes.

In this work, we have proposed a highly accurate model for energy assessment in WSNs, which can achieve all the goals previously mentioned. All the parameters that can affect the energy consumed in a WSN are considered by the presented mode. This has been illustrated by our experimental results that accurately model a real platform using the proposed model.

Bibliography

[1] Culler, D., Estrin, D., Srivastava, M. Overview of Sensor Networks. Computer, August 2004, 41-49.

[2] Paradiso, J.A., Starner, T. Energy scavenging for mobile and wireless electronics. IEEE Pervasive Computing, 2005, 18-27.

[3] Iwasaki, Y., Kawaguchi, N., Inagaki, Y. Design, Implementation and Evaluations of a Direction Based Service System for Both Indoor and Outdoor. Lecture Notes in Computer Science.

[4] Lo, B., Thiemjarus, S., King, R., Yang, G. Body Sensor Network– A Wireless Sensor Platform for Pervasive Healthcare Monitoring, Adjunct Proceedings of the 3rd International Conference on Pervasive Computing (PERVASIVE‘05), May 2005, 77-80.

[5] Feng, J., Koushanfar, F., Potkonjak, M. System-architectures for sensor networks issues, alternatives, and directions. IEEE ICCD, pp. 221-226, 2002.

[6] Bluetooth SIG, Inc. http://www.bluetooth.org

[7] Callaway, E., Gorday, P., Hester, L. Home Networking with IEEE 802.15.4: A Developing Standard for Low-Rate Wireless Personal Area Networks. IEEE Communications Magazine, August 2002.

[8] Ye, W., Heidemann, J. Medium Access Control in Wireless Sensor Networks. USC/ISI Technical report ISI-TR-580, October 2003.

[9] Mica mote. http://www.xbow.com

[10] Polastre, J., Szewczyk, R., Culler. D. Telos: Enabling Ultra-Low Power Wireless Research.

[11] Culler, D. TinyOS: Operating System Design for Wireless Sensor Networks, Sensors, May 2006.

[12] Levis, P., Lee, N., Welsh, M., Culler, D. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. ACM Sensys, 2003.

[13] Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E., Culler, D. The nesC language: A holistic approach to networked embedded systems. Proc. Programming Language Design and Implementation (PLDI), June 2003.

[14] Warneke, B., Last, M., Leibowitz, B., Pister, K.S.J. "Smart Dust: Communicating with a Cubic-Millimeter Computer", Computer, Jan. 2001. IEEE Computer Society, Piscataway, NJ. pp. 44-51.

[15] NEST project. http://nest.cs.berkeley.edu/

[16] BTNode. http://www.btnode.ethz.ch/

[17] DSYS25. http://www.cs.ucc.ie/misl/dsystems/HTML/dsys25.php

[18] ESB. http://www.scatterweb.com/

[19] ZebraNet project. http://www.princeton.edu/~mrm/zebranet.html

[20] Mainwaring, A., Polastre, J., Szewczyk, R., Culler, D., Anderson, J. Wireless sensor networks for habitat monitoring. ACM International Workshop on Wireless SensorNetworks and Applications (WSNA'02), Atlanta, GA, USA, Sept. 2002.

[21] CodeBlue project. http://www.eecs.harvard.edu/~mdw/proj/codeblue/

[22] Torfs, T., Sanders, S., Winters, C., Brebels, S., Van Hoof, C. Wireless network of autonomous environmental sensors. 3rd IEEE Conference on Sensors, 2004.

[23] Nordic nRF2401 Radio chip. http://www.nordicsemi.no

[24] Texas Instruments MSP430 Microcontroller. http://www.ti.com

[25] Microsoft Corporation. Microsoft Windows CE. http://www.microsoft.com/windowsce/embedded/

[26] Hildebrand, D. An Architectural Overview of QNX. http://www.qnx.com/literature/whitepapers/archoverview.html

[27] Palm, Inc. PalmOS Software 3.5 Overview. http://www.palm.com/devzone/docs/palmos35.html

[28] Wind River Systems, Inc. pSOSystem Datasheet. http://www.windriver.com/products/html/psosystem ds.html

[29] QNX Software Systems Ltd. QNX Neutrino Realtime OS. http://www.qnx.com/products/os/neutrino.html

[30] Microware. Microware OS-9. http://www.microware.com/ProductsServices/Technologies/os-91.html

[31] LynuxWorks. LynxOS 4.0 Real-Time Operating System. http://www.lynuxworks.com/

[32] Symbian. Symbian OS – the mobile operating system. http://www.symbian.com/

[33] uClinux Development Team. uClinux, The Linux/Microcontroller Project. http://www.uclinux.org/

[34] TinyOS 2.0. http://www.tinyos.net

[35] Dunkels, A. et al. Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. First IEEE Workshop on Embedded Networked Sensors, 2004.

[36] Han, C., Rengaswamy, R., Shea, R., Kohler, E., Srivastava, M. A dynamic operating system for sensor networks. Proceedings of the Third International Conference on Mobile Systems, Applications, And Services (Mobisys), 2005.

[37] Shnayder, V., Hempstead, M., Chen, B., Werner Allen, G., Welsh, M. Simulating the Power Consumption of Large-Scale Sensor Networks Applications. ACM Sensys, 2004.

[38] The Network Simulator – Ns-2. http://www.isi.edu/nsnam/ns

[39] Ousterhout, J.K. Tcl and the Tk Toolkit, Addison-Wesley, Reading, MA, USA, 1994.

[40] Chang, X. Network Simulations with OPNET.

[41] Fuster, V. Epidemic of Cardiovascular Disease and Stroke: The Three Main Challenges. Circulation, Vol. 99, Issue 9, March 1999, 1132-1137.

[42] Daskalopoulos, I., Diall, H., Raja, K. Power-efficient and Reliable MAC for Routing in Wireless Sensor Networks. Master's thesis, University College London, 2005.

[43] TinyOS Tutorial - Lesson 4: Composing components to Send and Receive Messages, http://www.tinyos.net/tinyos-1.x/doc/tutorial/lesson4.html, Sept 2003.