Sensor Cubes: A Modular, Ultra-Compact, Power-Aware Platform for Sensor Networks

K. Raja*, I. Daskalopoulos*, H. Diall*, S. Hailes*, T. Torfs**, C. Van Hoof** and G. Roussos***

* University College London, UK
{k.raja, i.daskalopoulos, d.diall, s.hailes}@cs.ucl.ac.uk
** IMEC, Integrated Systems Department, Belgium, {torfs, vanhoof}@imec.be
*** Birkbeck College, University of London, UK, g.roussos@bbk.ac.uk

ABSTRACT

The Sensor Cube platform is an ultra-compact, modular and power-aware way of building sensor networks; they are based on a stackable hardware design supported by a Tiny OS based operating environment. The Sensor Cube hardware measures 14x14x18mm³ and features an integrated coplanar antenna, a design that results in an ultra compact footprint. A core characteristic of the system is that its modular design allows for each of the radio, processor, sensing and power management layers to be interchangeable in a Lego-like manner. Moreover, its low power radio (based on the 2.4GHz Nordic nRF2401 design) and microcontroller (based on the Texas Instruments MSP430) allow for very efficient operation. The Sensor Cube operating and software development environment is derived from Tiny OS, which has been modified to meet the hardware requirements, in particular by introducing a power-aware and reliable ALOHA-type MAC protocol. In this paper we present our experience with the Sensor Cube platform and, in particular, the implications of its ultra-compact design on system performance, specifically as it relates to the characteristics and limitations of the radio unit.

Keywords: wireless sensor node, MAC protocol.

1 INTRODUCTION

In this paper, we report on the design, development a validation of a novel sensor network platform, the so-called Sensor Cube. Sensor Cubes are ultra-compact compared with the currently available sensor network platforms and provide for a modular hardware architecture which is supported by a software runtime derived from Tiny OS. In addition to the usual software development tools provided by the Tiny OS tool-chain, Sensor Cubes support a power-aware and reliable ALOHA-type MAC protocol that closely meets the characteristics of its radio unit. We also report on our experiences with this platform in the context of a series of empirical evaluation studies focused on the performance of standard multi-hop routing protocols.

This paper has the following structure: in the next section we discuss the main ingredients of the Sensor Cube platform with emphasis on the characteristics that set it apart from other existing sensor network platforms. In section 3, we briefly discuss the rationale for selecting Tiny OS as the foundation for developing the Sensor Cube runtime and in the following section we detail the challenges that had to be addressed in porting to the Sensor Cube hardware. Section 5 describes the design and development of a power-aware MAC protocol that closely meets the operational capabilities of the wireless component and, in section 6, we discuss its performance in specific case studies. We conclude with a brief discussion of our findings.

2 HARDWARE PLATFORM

Sensor Cubes are built on the hardware platform recently developed at IMEC [1, 11] which offers two distinct advantages over the current state-of-the-art: first, it provides for an ultra-compact design including an integrated coplanar antenna that allows for very low power consumption; and second, it supports pluggable modules that allow for the physical reconfiguration of nodes to include only the functionality required for a particular application. The combination of these two characteristics implies that Sensor Cubes are versatile enough to support a variety of application scenarios within the same modular design:

- In cases where a large geographic area must be covered with a low density sensor network a more powerful radio could replace the less powerful short range radio that is more suitable for indoor or body sensor network.
- In cases where high data rates and complex signal processing functions are required, a more powerful digital signal processor could be used.
- In cases where specific specialized sensors and associated sensor electronics are required (for example chemical or biosensors) they could be accommodated within this design on a separate module.
- In cases where power beyond that provided by the battery is necessary, or when a full power management system with scavenging is needed, such components could also be developed and added as separate modules.

In this section we will discuss in turn the currently implemented modules, including processing, wireless and sensing modules.

2.1 Overall Cube Architecture

The currently available hardware modules of the Sensor Cube platform include the microcontroller, radio communication, power and sensor. The prototype implementation features these four functional blocks, each

14x14mm2 in size printed circuit boards plugged together to make up a four-layer stack (cf. Figure 1). The stacked implementation using connectors was 18mm high. In an alternative implementation of this design solder-ball interconnections are used instead of connectors [1], thus reducing the height of a single node to only 10mm or less, depending on the application layers included (cf. Figure 2).

At the top of the stack is the Radio Layer, mostly occupied by a Nordic nRF2401 2.4GHz wireless transceiver chip [6], together with an integrated antenna. The second layer incorporates the Texas Instruments MSP430 microcontroller [9] which is the "heart" of the platform as it is responsible for data processing and control. In the same layer, a 32.768kHz crystal provides a local time reference and a clock source to the system. This layer also provides the following features:

- Digital input/output
- 12-bit analog to digital converter (built into the MSP430)
- Universal synchronous-asynchronous receive/transmit
- Clock System and Timers

The microcontroller and radio layers together form the core of the Sensor Cube platform and they are designed to work closely together, a fact that justifies their physical proximity – direct connection.

Below the Microcontroller Layer, two additional layers provide the Power Management (layer 3) and Sensing (layer 4) features. The Power Management layer is designed in such a way that it can accept power from an energy harvesting device (including, but not limited to, solar cells and vibration scavengers) so as to sustain the battery life. A standard battery is also connected to this layer: for example, the Varta 2-cell NiMH batteries, with a nominal voltage of 2.4V. The available sensing equipment is comprised of a Sensirion SHT15 Temperature/Humidity sensor and a Light-Dependent Resistor (for measuring illumination). These commercial-off-the-shelf sensors produce accurate measurements while consuming very little power when in use or standby.

2.2 Nordic nRF2401 Radio Transceiver

Of particular relevance to this work are the characteristics of the radio module and, for this reason, we will address it in more detail in this section. The radio transceiver chip provides all the hardware necessary for transmitting and receiving at the 2.4-2.5GHz ISM band in a tiny package and is characterized by its low power consumption, built-in power-saving modes and relatively high bit-rates for transmission/reception (250kbps and 1Mbps). Control and configuration of the Radio is achieved by loading to it a 15-byte configuration word (or parts of it, depending on the changes required). The most notable feature of the Nordic radio chip is its ability to transmit and receive data in two different modes: the ShockBurst Mode and the Direct Mode. The very useful DuoCeiver feature allows simultaneous reception of two different signals, provided that they are 8MHz apart. This means that, even though a single antenna is used, the Radio can receive simultaneously from two potential transmitters – in other words, it has two channels. When in ShockBurst mode, each of the channels can have its own address that can range from 8 to 40 bits. In Direct Mode it is the responsibility of the software to carry out any address processing.

Finally, the Nordic nRF2401 supports the following modes of operation (power characteristics shown in parentheses):

- Transmit Mode (13mA average at 0dBm output Power)
- Receive Mode (23mA average for both channels on)
- Configuration Mode (12uA average)
- Stand-By Mode. (12uA average)
- Power Down Mode (400nA Average)

2.3 Sensing Components

As noted earlier, for the purpose of humidity and temperature measurement the Sensor Cubes are equipped with the Sensirion SHT 15 [8] multi-sensor module. The SHT15 can be configured to measure either relative humidity with an accuracy of ±2% RH or temperature with an accuracy of ±0.5ºC, by configuring a digital measurement mode register. The SHT15 sensor includes a 14-bit analog to digital converter. It provides calibrated, digitized data to the microcontroller via a serial interface. In addition, the lower sensor layer of a Sensor Cube contains a cadmium sulfide Light-Dependent Resistor (LDR) to measure illumination (in lux, after calibration with accuracy ±3%). This component is connected to one of the eight ports of the 12-bit ADC built into the MSP430.

This Sensor Layer based on commercial-off-the-shelf sensors, completes the Sensor Cube as an environmental sensing module. For other applications, different custom or off-the-shelf sensors can be used with the Sensor Cube to meet the requirements of the application, thanks to the modularity and pluggability of the cube design.

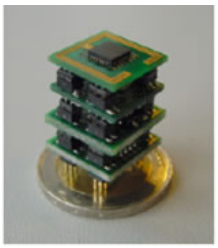

Fig. 1 The four layers of the IMEC Sensor Cube hardware platform. A 2€ coin provides a size reference.

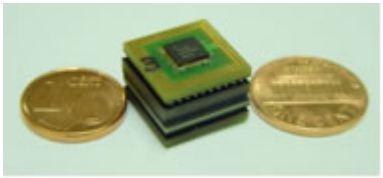

Fig. 2 The Sensor Cube alternative hardware implementation using solder ball interconnect technology.

3 SOFTWARE PLATFORM

In this section, we briefly discuss the rationale for the selection of Tiny OS as the foundation of the software development component of the Sensor Cube platform. To this end, the objectives of this work have been twofold: to identify a system that offers a suitable foundation in terms of software components and tools for further development with the Sensor Cube hardware, while at the same time restricting the amount of work required for developing the new components. Several alternatives were considered, including the standard embedded systems development approach of low level libraries that provide a skeleton functionality; and porting one of a number of emerging sensor network operating systems including Tiny OS [5], Contiki OS [2] and the Sensor Operating System [4]. We selected TinyOS as it offers several advantages:

- It is one of the more mature sensor network platforms and provides a rich collection of development tools, routing protocol implementations and applications
- Tiny OS supports the MSP430 microcontroller and a hardware abstraction layer is readily available
- Tiny OS provides a simulator that uses a probabilistic bit error model which is relatively scalable

The first step towards supporting Tiny OS on the Sensor Cubes related to the development of those components that supported the operation of the Nordic radio. However, it soon became apparent that simply supporting the standard Tiny OS MAC protocols was inefficient for the Nordic device as they failed to take advantage of its particular capabilities. As a consequence, we proceeded to design and implement an alternative.

4 TINY OS FOR SENSOR CUBES

Tiny OS has a simple mode of operation, following an event driven paradigm: every processing step is triggered by an event of some type and tasks triggered as a result of such events are added to a worker queue that is processed by the main program loop. This approach allows for a similarly simple system building process since the full operating system and the application are built together in one step. The nesC [3] compiler combines all components of the application and operating system and builds a C file that is passed to the compiler. Thus, supporting a new platform with Tiny OS involves modifying the following sections of the source code tree:

- the platform independent operating system part (system),
- common interface definitions (interfaces),
- a library with commonly used functions (lib),
- platform dependent hardware definitions and access driver functions (platform), and
- definitions for different sensor boards that can be used in combination with the motes (sensorboard).

After the binary is prepared and uploaded to the sensor node, the program loads in memory and executes --- a process that combines system and application specific procedures. Initialization of the system components is carried out by the so-called Bootup process, which always starts at Main.nc with its first call to hardwareinit(). Subsequently, this function calls TOSH SET PIN DIRECTIONS() from hardware.h, which in turn calls macros to set the direction registers of the microcontroller and initialize the hardware clock. Finally, the Std-Control.init() and start() functions are called and dispatched to all connected modules. After enabling interrupts, the Tiny OS system kernel enters an infinite loop calling TOSH run task() repeatedly. This function processes all pending tasks in the queue until none is left and then enters sleep mode until an interrupt request (IRQ) occurs. For a new platform, two files are required to describe its architecture, namely .platform and hardware.h. The former is described in detail in the following section and the later is used to assign functions to the pins of the microcontroller.

4.1 Setting up the Build Tool-Chain

The Tiny OS build tool-chain consists of the make system, the nesC compiler driver and nesC-to-C preprocessor. The make system provides simple ways to compile install and otherwise manage Tiny OS programs for different platforms, using a set of standard options. To do this, it invokes ncc, the driver for the compiler, and other software packaging and installation tools including uisp and the TI loader msp430-bsl. This system is implemented as an overlay to the GNU software development tools and compilers and, in fact, ncc is simply a pre-processor for nesC, with the C compiler responsible for most of the actual work. This has significant implications for debugging as variable and function names appear in the debugger output with their C rather than nesC names.

Specifying the build options so that the tool-chain will produce correct code for the new platform is primarily the role of the .platform file. Several more configuration files for the tool-chain are required:

- .target files (valid make target)
- .extra files (dummy target for defining extra make variables)
- .rules files (part of “msp” subdirectory)

Thus, the new platform name (imec) was defined, all necessary files specifying the different aspects of the compilation process for this platform had to be created and added to the source code tree.

4.2 Establishing Radio Communication

To simplify management of system development processes, Tiny OS defines standard within the Tiny OS Extension Proposals (TEP) [10]. Two TEPs deal with radio

code: Radio Physical Layer and Radio Link Layer, which provide the hardware abstraction architecture for Radio Components implemented by Tiny OS. The Hardware Abstraction Architecture (HAA) for radio components is split in three separate layers: the Hardware Physical Layer (HPL), Hardware Adaptation Layer (HAL), and Hardware Interface Layer (HIL).

As the basis for our implementation of the Sensor Cube HAA components, we used software developed by DSYS25 [7], a sensor platform based on the Atmel AVR ATMEGA microcontroller and the Nordic nRF2401. In particular, we re-engineered a large part of the radio transceiver code as appropriate for our platform.

The correct implementation of the HPL and HAL and the actual radio communication layers was verified by testing standard Tiny OS applications.

4.3 Supporting the Sensors

As noted earlier, the Sensor Cube prototype implementation provides three sensing devices built into the Sensing Module and a further temperature sensor built into the MSP microcontroller. Supporting these sensors in Tiny OS involved the following steps.

Sensirion SHT15 Humidity and Temperature Sensor. The Sensirion SHT11 sensor has been used in the Telos platform and thus a suitable driver is readily available in the Tiny OS source code. The SHT11 employs a one-wire protocol which is similar to the SHT15 but the latter carries out its own analog to digital conversions. The SHT15 components which perform this conversion support the standard ADC and ADCError interfaces of Tiny OS and thus the implementation of a suitable driver was relatively straightforward. This design proved robust in testing with no problems encountered. Moreover, it provides a consistent external view of the sensor to other Tiny OS components and is compliant to the specifications in TEP 101: Analog-to-Digital Converters [10]. Finally, the appropriate pin names and settings had to be configured in hardware.h and modifications to the interrupt pin to P1.1 were made to match the SHT15 specification.

Light-dependent Resistor (LDR). Support for the 12-bit ADC incorporated in the MSP430 is fairly mature in Tiny OS and follows closely the hardware abstractions specified in TEP 101. Thus, enabling the LDR of the Sensor Cube involved a simple modification of the existing code base to read the LDR output on input channel 0 (pin P6.0, configured as peripheral function).

MSP430 Internal Temperature Sensor. This internal sensor is connected to input channel 10 on the ADC and is directly supported in the MSP430 platform.

5 MAC PROTOCOL DESIGN

Although assembling a fully functioning system has been a critical milestone for the Sensor Cube platform, in practice it became evident from early on that it was necessary to design an optimized protocol stack. This task involved the development of an energy-efficient MAC layer that closely fits the capabilities of the radio component. In this section, we detail the design of this protocol stack.

There are several limitations imposed by the characteristics of the Nordic design: due to lack of a high speed clock source, which would lead to substantially higher power consumption, it is not possible to employ the transceiver's Direct Mode, which would allow for better radio control. Moreover, in Shockburst mode, the control header and the payload cannot exceed 32 bytes, which is the maximum frame size. Finally, the radio configuration word cannot be altered while the device is transmitting or receiving data.

5.1 MAC Design Overview

Two choices were available to us in transmitting acknowledgement packets:

• Use the same channel for ACKs as for the data. However, ACKs are shorter than data packets, so either we simply fit an ACK within the larger data packet and transmit more than is necessary (wasting energy), or we use the data width field in the configuration word of the Nordic interface to alter the size of the Shockburst frame between ACK and data packets. Unfortunately, the time required to carry out this change is 1ms, which is approximately equal to the time required to transmit a full frame at 250kbps.

• Recall that the radio component can be operated in two separate channels, and, as we elected, to use Channel 1 to send data (with a full frame size of 32 bytes) and Channel 2 for sending/receiving ACK packets (with a frame size of 13 bytes).

The maximum ShockBurst frame of 32 bytes includes the ShockBurst address, the CRC and the payload requested by the MCU. In our implementation, the payload represents an Active Message (AM) packet constructed by the associated TinyOS layer, and contains control header fields and actual data sent from the application. Inevitably, the question of fragmentation arises as a result of the limited Shockburst frame size and the fact that the overhead incurred by the different components allow only for 20 bytes of payload available to the application (3 bytes are used for Shockburst address, 1 byte for CRC and 8 bytes for AM control header). Nevertheless, taking into account the actual data that is transmitted in a sensor network under normal operating conditions, we decided not to address fragmentation at this stage: applications normally would send only a few bytes of data due to sensor readings and some amount of control information as part of the routing protocol. In this scenario, providing for fragmentation at this level would mostly reduce the efficiency of the system as it would add additional header data which would be unnecessary for the vast majority of packets. Instead, those

applications that require fragmentation are expected to add it as needed.

The use of the Shockburst mode as the principal mode of communications implies that two addresses are encapsulated in every data packet: the Shockburst broadcast address and a node-specific address within the AM header. This approach allows us to combine the performance advantages of Shockburst while at the same time retaining the capability to address data to specific nodes and thus maintaining unicast semantics.

The next design decision was for the receiver to use a set duty cycle that switches the radio between stand-by and receive mode at regular intervals, in order to reduce the overall level of energy consumed. The amount of time the radio stays in either mode is configurable at application development time, and a more detailed discussion of the implications of various choices for duty cycle can be found in section VI.

For the same reason, we decided against using carrier sense and, rather, elected to use a simple Aloha-based protocol instead. In the case of carrier sense, on transmission, the radio listens to the channel to determine if it is used by some other station. However, if collisions are rare, either as a result of low node density or low transmission rates, as we expect for these nodes, then the higher energy cost required by this approach is not justified by a requirement for collision avoidance.

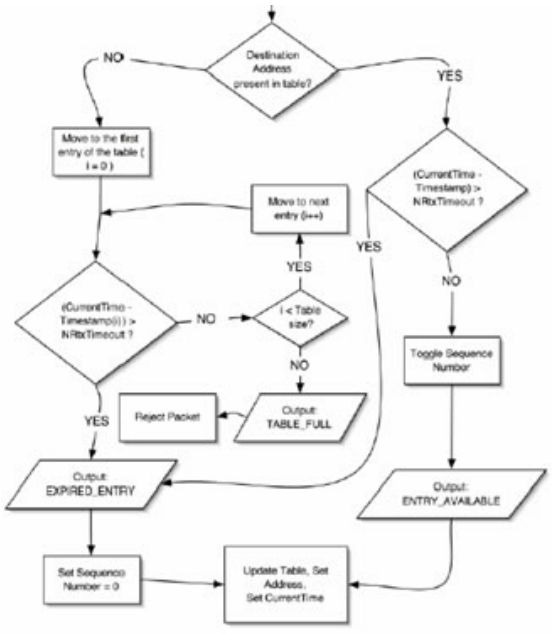

Fig.3. Flowchart describing the logic of the Transmitter's MAC table.

Nevertheless, it is still necessary to address the problem of collisions whenever they arise. A further complication of the selected solution is the fact that a receiver node may be in stand-by mode when the transmitter is sending data and hence unable to receive them. To address both of these problems, we employ link level acknowledgements and retransmissions. In this scheme, as soon as the transmitter sends data, it waits for an acknowledgement for a set duration of time, which is also configurable. If no acknowledgement arrives within this time frame, then the packet is retransmitted. Note that since the transmitter cannot distinguish a packet which was lost due to a collision or because the receiver was in stand-by mode, the transmitter resends its packets until it receives an acknowledgement or reaches the Maximum Retransmission count (which is also configurable in application code).

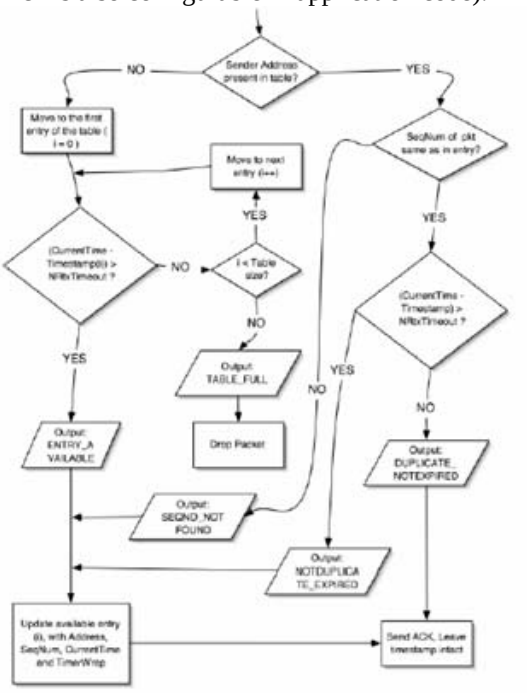

Fig 4. Flowchart describing the logic of the Receiver's MAC table.

A further technique that we considered for inclusion in the protocol, was RTS/CTS (Request to send/Clear to send), which is a mechanism often used in wireless systems to avoid the hidden and exposed terminal problems. To avoid both of these problems, the RTS/CTS mechanism is used to establish a session before initiating the transmission thus removing the possibility of collision from data packets, which can be long, to RTS/CTS packets, which are short and therefore less likely to result in collision. In our case, data packets are themselves not long, and so the value of adding the RTS/CTS mechanism is at best questionable. The hop-by-hop data forwarding which uses a unicast transmission is protected by the link-level acknowledgements which we introduced and

retransmissions, and effectively addresses the hidden terminal problem.

5.2 MAC with Idle-RQ Table Management

The MAC layer can have no more than one outstanding packet at any given point of time. A packet is said to be outstanding, when it has been transmitted and not yet acknowledged and the maximum retransmission count has not yet been reached. Allowing multiple outstanding packets might result in a collision (with any ALOHA based approach) and will also increase the complexity of sequence number management. Before performing the actual transmission of the packet, the following checks will be performed by the MacTable as described in the flowchart in Figure 3. A similar table is maintained at the receiver and its operation is highlighted in Figure 4.

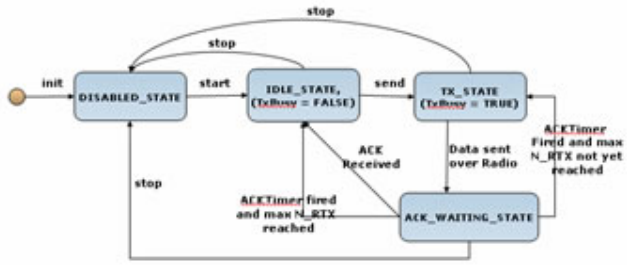

Fig.5. Radio State Diagram

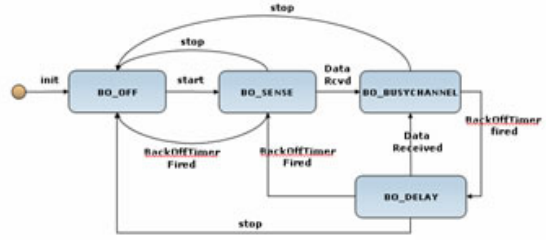

Fig.6. Backoff State Diagram

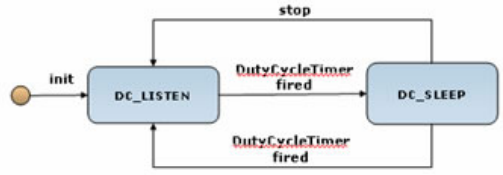

Fig.7. DutyCycle state diagram.

Figure 5-7 show the the various states and events that cause state transitions within the MAC layer. In summary, the Sensor Cube MAC provides no fragmentation, thus eliminating the fragment header overhead; does not employ no carrier sense; and implements a low data width Shockburst frame for acknowledgements.

6 EVALUATION AND PERFORMANCE ANALYSIS

To evaluate the performance of the Sensor Cube platform a number of experiments were carried out. In this section we explore the behavior of the proposed MAC protocol in the context of a controlled environment. We aim to draw conclusions regarding the effects of different choices of parametres regarding the duty cycle and retransmission on the performance of the MAC protocol. The evaluation process aimed specifically to provide an understanding of the trade-offs involved and draw conclusions on how different duty cycle lengths and ratios, acknowledgements and retransmissions can affect packet delivery ratio. The aim has been to identify the optimal set of parametres for some application. Rather than have this discussion in the abstract, we specifically considered the case of data harvesting using the Surge application [5], a simple tree-based acquisitional query engine available with Tiny OS.

6.1 Experimental Setup

The experiments were conducted in an environment that supports several wireless LANs, and as a consequence there was a significant amount of interference, as both Sensor Cubes and the IEEE 802.11 protocol use the 2.4GHz band. We did not investigate the case where physical obstacles were located between the communicating nodes, as they were found to communicate with great difficulty even through a single wall. For the experiments, two Sensor Cubes were used that were powered from constant voltage sources in order to eliminate any effects that inadequate power supply would introduce. The first sensor was powered by a laboratory power supply, set to output 2.7V, whereas the second was powered directly from the USB programming board that also provided 2.7V. The sensors were placed in positions which allowed line of sight and at distances of 1, 6 and 12 metres. Since the aim was to explore the impact of acknowledgements, retransmissions, physical proximity and duty cycle lengths and ratios to packet delivery ratio, a driver application was developed to enable the exploration of the parametre space.

For testing, a two part application was used with a Transmitter and a Receiver role. The transmitter sent 100 packets at a time. In the case where acknowledgements were used, the timeout was set to 4ms. After 8 retransmissions had been attempted the transmitter moved on to the next packet. Thus, a transmitter could potentially send up to 800 packets if no acknowledgements were ever received. In either case, packets were sent at regular intervals which, on success, were 175ms.

On the receiver side, the overhead of writing data to the USART so as to record on the laptop host was found to be significant and interfered with the operation of the protocol.

Therefore, it was decided to minimize communication to and from the laptop by maintaining the received packet count in a dedicated variable in the receiving Sensor Cube's memory, the value of which was not transmitted until the end of the packet transmission cycle. Since the number of packets could vary, a watchdog timer was used to flag an upper threshold. The timer itself did not affect the performance of the protocol. The experiments conducted involved measurements of both the transmission and reception sides.

6.2 Experiment Analysis

Two sets of experiments were conducted. In the first case, acknowledgements and retransmissions were disabled in order to give a baseline measurement. The transmitter and receiver were placed 6 metres apart and 100 Shockburst packets were transmitted, with different duty cycle periods at the receiver. For each duty cycle value, the experiment was repeated three times and the average packet delivery ratio was recorded. The results are shown in Figure 8.

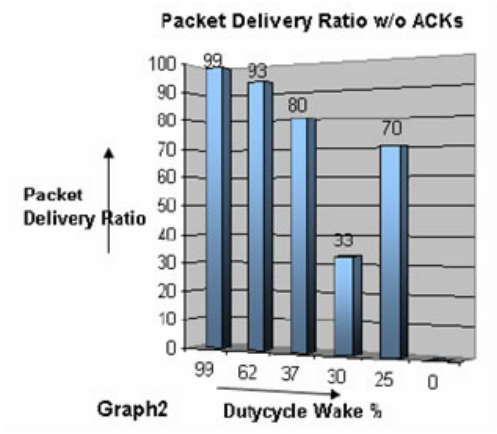

Fig. 8. Packet Delivery ratio with disabled acknowledgements

From the graph of Figure 10 can be seen that a 62% duty cycle wake time the packet delivery ratio seems to be relatively high (93%), but if the wake time is decreased, the packet delivery ratio is reduced before it rises again. The intermediate reduction is the result of a misalignment between transmission periods and reception periods, and the increase simply a synchronization effect. Hence, to achieve a packet delivery ratio of at least 90%, the receiver must have a duty cycle of no less than 60% wake time (and thus 40% sleep time).

In the second set of experiments, the same setup was used as for the first set;. however, acknowledgements and retransmissions were enabled. From Figure 9, it is clear that, in spite of very low wake time in the duty cycle (25% wake time) the packet delivery ratio remained high at 96%. Thus, this version of the MAC protocol smoothes out the synchronization effects seen in the previous experiment, in addition to providing reliable delivery. At first glance, this approach also appears to save energy, since the receiving node can remain off for a greater proportion of the runtime. However, reality is not quite so simple: acknowledgements and retransmissions also require energy for both parties as both nodes must be enabled, though only for the relatively short time window in which an ACK is generated. Likewise, to raise the packet delivery ratio from the one shown in Figure 8 to that of Figure 9, retransmissions have clearly happened. In fact, for a 30% duty cycle, around three retransmissions will be needed to raise delivery rate from 33% to 97%. Although, such communication also has a considerable energy cost associated with it, this cost is borne by the transmitter rather than the receiver. Thus, the addition of ACKs will tend to move energy consumption away from the receiver and towards the transmitter.

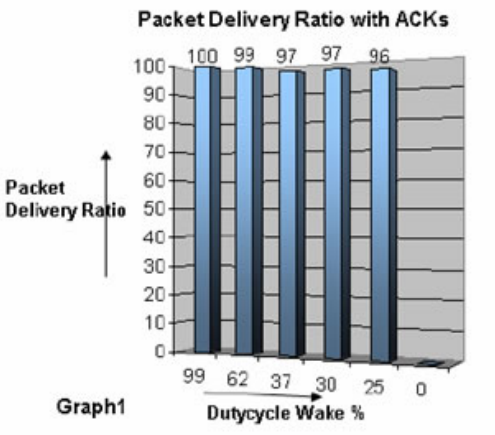

Fig 9. Packet Delivery ratio with ACKs enabled.

The true picture of energy costs requires careful consideration of the number of retransmissions needed for a packet to be received at a given duty cycle. However, this is also not simple, for the same reasons of synchronization that lead to the non-monotonicity of the first experiment. Thus, a third set of experiments were conducted, again to observe packet delivery ratios, but this time by choosing different sleep/wake cycle durations at the receiver, though all with the same proportion of wake time (37.2%) to sleep time (62.5%) and without ACKs enabled. The recorded values are plotted in Figure 10. Even with the same duty cycle, synchronization effects between sending and receiving windows cause widely varying delivery ratios and, consequently, widely different energy consumption figures.

Further work is required in this area. Nevertheless, it is our contention that if transmissions are irregular, then the inherent randomness in the synchronization would tend to avoid a situation of persistently low delivery ratio. If, on the other hand, the transmissions are regular, as in the experiments above, various alternative approaches to increasing delivery rates should be considered. An initial

phase in which ACK and retransmission occurred could be used to understand how accurate was the synchronization (provided that the receiver knew that a packet were a retransmission, something that would cost an extra bit), and thus drive the receiver towards synchronization with the transmitter. The situation is more complex in the case of multiple transmitters, but the process can be repeated for different receive windows. All of these have impacts on the energy consumed, leading to a rather complex picture.

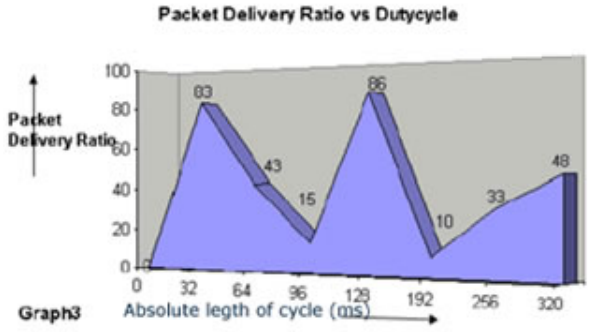

Fig.10. Packet Delivery ratio with different cycle durations (in ms). Duty cycle is 37.5% awake in one contiguous period.

Finally, it must be noted that these experiments are intimately related to the application in question. Thus, in realistic deployment scenarios, a fine tuning step would be required to identify the optimal wake up/sleep cycle which best balances data quality requirements and energy consumption.

7 CONCLUSIONS

The Sensor Cube is a new sensor network platform based on IMEC's ultra-compact, modular sensor hardware. The modularity of the design allows for increased flexibility in tailoring node capabilities to the application at hand and also provides a coplanar antenna. A software development environment based on Tiny OS complements the hardware thus providing a complete platform for the development of sensor network systems. To this end, a simple, reliable, ALOHA based, power efficient MAC protocol with appropriate duty cycle management has been introduced to closely meet the requirement and limitations of the hardware. Empirical evidence suggests a high packet delivery ratio (above 95%) with relatively low radio duty cycles (25% active). Multi-hop routing protocols bundled with Tiny OS have been shown to operate effectively on top of the Sensor Cube radio stack, and were tested experimentally and in simulations.

REFERENCES

[1] K. Baert, B. Gyselinckx, T. Torfs, V. Leonov, F. Yazicioglu, S. Brebels, S. Donnay, J. Vanfleteren, C. Van Hoof, "Technologies for highly miniaturized autonomous sensor networks", First International Workshop on Advances in Sensors and Interfaces (IWASI), Bari, Italy, April, 2005.

[2] A. Dunkels, B. Gronvall and T. Voigt, "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors", First IEEE Workshop on Embedded Networked Sensors (EMNETS-I), Tampa, Florida, USA, 16 November, 2004.

[3] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesC Language: A Holistic Approach to Networked Embedded Systems", Proc. Programming Language Design and Implementation (PLDI) 2003, San Diego, California, USA, 8-11 June, 2003.

[4] C. Han, R. Rengaswamy, R. Shea, E. Kohler, and M. Srivastava, "A dynamic operating system for sensor networks", Proc. of the Third International Conference on Mobile Systems, Applications, and Services (MOBISYS), Seattle, Washington, USA, 6-9 June, 2005.

[5]P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer and D. Culler, "The Emergence of Networking Abstractions and Techniques in TinyOS," Proc. First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2004), San Fransisco, California, USA, 29-31 March, 2004.

[6] Nordic Semiconductor, "nRF2401A Single Chip 2.4 GHz Radio Transceiver", Available online at http://www.nordicsemi.no/files/Product/data_sheet/nRF2401A_rev1_0.pdf..

[7] B. O'Flynn, A. Barroso, S. Bellis, J. Benson, U. Roedig, K. Delaney, J. Barton, C. Sreenan, and S. O'Mathuna. "The Development of a Novel Miniaturized Modular Platform for Wireless Sensor Networks." Proc. IPSN Track on Sensor Platform, Tools and Design Methods for Networked Embedded Systems (IPSN2005/SPOTS2005), Los Angeles, USA, IEEE Computer Society Press, April 2005.

[8] Sensirion, "SHT1x/SHT7x Humidity & Temperature Sensor", Available online at http://www.sensirion.com/images/getFile?id=25.

[9] Texas Instruments, " MSP430x1xx Family User's Guide", Available online at http://ti.com/msp430

[10] Tiny OS 2.0 Working Groups. Documents available at: http://www.tinyos.net/scoop/special/working_group_tinyos_2-0

[11] T. Torfs, C. Van Hoof, S. Sanders, C. Winters, and S. Brebels, "Wireless network of autonomous environmental sensors", IEEE Sensors Conference, Vienna, Austria, April, 2004.