Idleness as a Resource in Energy-Neutral WSNs

Alessandro Bogliolo, Emanuele Lattanzi, Valerio Freschi
Department of Basic Sciences and Foundations
University of Urbino
{alessandro.bogliolo, emanuele.lattanzi, valerio.freschi}@uniurb.it

ABSTRACT

In spite of the availability of ultra-low-power microcontrollers and radio transceivers, the power consumption of an active sensor node is much higher than the power provided by state-of-the-art harvesters of suitable size and cost. Hence, the feasibility of energy-neutral wireless sensor networks mainly depends on the capability of the nodes to exploit idle periods to recover the energy spent to perform the tasks assigned to them. This paper discusses the main issues which prevent WSNs to fully exploit the idleness and presents a general power state model capturing the energy efficiency of a mote. VirtualSense motes are used as case study to characterize the proposed power state model and to illustrate its application.

Categories and Subject Descriptors

C.2.1 [**Computer-communication networks**]: Network Architecture and Design- *Wireless communication*

General Terms

Performance, Design, Measurements

1. INTRODUCTION

The combination of energy harvesting technologies with ultra-low-power design practices and advanced dynamic power management techniques makes it possible to conceive energy-neutral wireless sensor networks (WSNs) the lifetime of which is not battery constrained. The design for unlimited lifetime imposes a paradigm shift from energy-constrained lifetime maximization, typical of battery-operated devices, to power-constrained workload maximization, suitable for energy harvesting systems. Given the environmental conditions and the task assigned to a node, the amount of workload it can sustain with the power scavenged from the environment depends on the efficiency of the harvester and on the energy efficiency of the node. State-of-the-art ultra-low-power microcontrollers exhibit 16-bit [8] or 32-bit [16] CPUs clocked at tens of MHz, which provide a good tradeoff between power and performance, while also providing many inactive modes which offer a valuable support to dynamic power management (DPM). In fact, in spite of the energy efficiency of the microcontrollers, the power consumption of an active sensor node is much higher than the power provided by state-of-the-art harvesters of suitable size and cost. Hence, the feasibility of energy-neutral wireless sensor networks strongly depends on the possibility of exploiting idle periods to turn most of the components and recover the energy spent during the active periods.

ENSSys'13, November 13 2013, Roma, Italy

http://dx.doi.org/10.1145/2534208.2534214

In practice, the operating conditions in which each node operates are determined by two external elements: the environment and the workload. Wireless sensor nodes have to exploit both environmental energy and workload idleness in order to achieve the energetic neutrality. As energy harvesters are used to scavenge energy from the environment, DPM techniques are used to exploit the idle periods of the workload. The energy efficiency of a node depends on the efficiency of both its harvester and its DPM support.

This paper provides an overview of the main issues which prevent WSNs to fully exploit the idleness of the workload and introduces a general power state model that captures the energy efficiency of a mote from a DPM perspective. The proposed model is applied to a representative case study, based on an open-hardware ultra-low-power mote.

2. EXPLOITING IDLENESS IN WSNS

This section discusses the exploitability of the idleness of the workload of a general purpose wireless sensor node, focusing on three main aspects: the hardware support to DPM offered by ultra-low-power microcontroller units (MCUs), the overhead introduced by the software stack, and the issues raised by the communication needs.

2.1 Microcontroller Unit

Ultra-low-power MCUs [8, 16] provide an energy efficient active state and several inactive low-power modes (LPMs). The first way of exploiting the degrees of freedom offered by the workload is to slow-down the processor up to the limits possibly imposed by application-level real-time constraints. This can be done by adjusting the clock frequency and by scaling the supply voltage accordingly within the ranges admitted by the MCU family adopted. Frequency and voltage levels can be adjusted at design time, at configuration time, or at run time. In the latter two cases, the flexibility offered by dynamic voltage scaling is partially paid by the

limited efficiency of the integrated dc-dc converter, which looses part of the benefits that could be achieved by providing directly the minimum supply voltage needed to sustain the clock frequency of choice. As for frequency scaling, its main advantage is that of enabling voltage scaling. In fact, slowing down the clock without reducing the supply voltage would not reduce the overall energy spent to execute a task. Thanks to frequency/voltage scaling, the mode of a given MCU provides an entire range of operating conditions the power and performance of which depend on the clock frequency (f) and voltage level (Vcc) adopted.

When the MCU has no tasks to execute and no input events to process, it can be placed in an inactive mode in which the CPU is turned off to save power. The LPMs provided by typical MCUs can be classified into three broad categories, hereafter denoted by the following names: *Standby*, in which the CPU is turned off, but the clock and memory systems are still powered; *Sleep*, in which the internal clock system is turned off so that the MCU looses the capability of issuing self wakeup events and measuring the time elapsed since last shut-down; and *Hibernation*, in which even the memory system is turned off, so that there is no data retention. The three LPM categories provide different tradeoffs between power consumption, wakeup cost, and functionality. Multiple power states could be available within each category depending on the technical solutions adopted by the MCU either to improve energy efficiency or to mitigate the drawbacks of the LPMs. For instance, an external real-time clock (RTC) could be used to provide time awareness and trigger self-wakeups from Sleep and Hibernation modes, while non-volatile memories could be used to preserve state information in Hibernation mode.

2.2 Software Stack

In principle, the power modes of the MCU could be directly exploited by ad-hoc bare-metal applications running without any operating system (OS). Productivity, portability, and maintenance issues, however, have motivated the development of lightweight operating systems [13, 2] and virtual machines (VMs) [12, 17] suitable to be executed by ultra-low-power MCUs. As a matter of fact, the most common platforms for WSNs exhibit a tiny OS and, possibly, a virtual runtime environment. This section discusses the overhead introduced by the software stack and its impact on DPM.

The basic DPM mechanism provided by an OS for WSNs (e.g., Contiki [2]) exploits the Standby mode of the MCU whenever all the running processes are waiting for scheduled timers or external events. A timer interrupt is used in Standby mode to turn on the CPU periodically in order to check for elapsed wakeup times. The period of the timer interrupt (T) is a constant defined at compile time and initialized once and for all during the boot. The value of T (which is typically 10ms) determines the time resolution of the events managed by the OS, including asynchronous external interrupts that are treated by the OS as if they were aligned with the last timer interrupt. Sleep and Hibernation modes are not exploited by default because of the lack of timing information and data retention, which would impair the functioning of the OS. These deeper LPMs, in fact, could be directly exploited only in case of time-independent, memory-less tasks triggered by external events.

The OS adds to the cost of wakeup both in terms of energy and in terms of time. Wakeups from Standby and Sleep states entail the execution of the interrupt handler and, possibly, of the OS scheduler before eventually resuming the execution of the task, while wakeups from Hibernation also impose a complete reboot of the OS. The situation is even worse when an additional layer is added to the software stack to provide a virtual runtime environment. Consider, for instance, a virtual machine like the Darjeeling VM [1], which executes as a single process on top of the OS. The VM needs to resume at each timer interrupt in order to check for the status of its threads. If there are no threads ready to resume, the process is suspended until next timer interrupt. Hence, the scheduler of the VM needs to be executed at each wakeup before either resuming the task or deciding to go back to the original LPM. In case of wakeup from Hibernation, the VM needs to be rebooted from scratch right after the OS boot.

In a typical scenario, the execution of the Darjeeling VM on top of Contiki would preclude completely the use of LPMs, making even the wakeup time from Standby longer than the wakeup period ($T = 10ms$) needed by the OS to maintain time awareness. In principle, however, periodic wakeups could be avoided if the MCU was able to react to external events and there was an oracle able to turn on the CPU right in time to execute the tasks. Both mechanisms exist in any mote, in that asynchronous external events can trigger hardware interrupts, while the scheduler of the OS has the capability of acting as an omniscient oracle for the self-events scheduled by its processes. Similarly, the scheduler of the VM can act as an oracle for the self-events scheduled by its threads. Hence, the LPMs of the MCU can be effectively exploited in spite of the software stack by dynamically adjusting the wakeup period (T) according to the scheduled events, while possibly using a low-power RTC in order not to impair time accuracy [10].

2.3 Communication

WSNs make use of multi-hop routing schemes (with star, mesh, or hybrid topology) to enable a bidirectional communication between each sensor node and a central node acting as a gateway (usually called *sink*). Most protocols entail a *dissemination* phase, in which control messages are diffused by the sink to assign a monitoring task to each node and to update routing tables, and a *collection* phase, in which sensed data are sent from the involved sensors to the sink. As a matter of fact, communication across the radio channel is responsible for most of the power budget of the motes. Three main communication issues are discussed in the following: the power spent for listening, the lack of synchronization, and the energy waste caused by the reception of unintended packets.

2.3.1 Listening

In order to be ready to receive control messages and possibly route sensed data, all the nodes of a WSN have to listen to the radio channel by keeping the radio transceiver in reception mode, with an average power consumption of tens of mW, which is several orders of magnitude higher than that of the MCU. While there are no ultimate solutions for the reducing the listening power consumption [18], state-of-the-art ultra-low-power radio transceivers (e.g., TI CC2520 [7])

provide LPMs in which the radio module is turned off to bring the consumption below 1mW, or even below 1μW at the cost of a longer wakeup time. The exploitation of the LPMs of the transceiver, however, impairs the capability of the node to receive incoming messages, giving rise to global synchronization issues.

2.3.2 Synchronization

In order to apply DPM techniques to the radio transceivers of the sensor nodes without impairing their communication capabilities, it is necessary to guarantee that, whenever a packet has to be exchanged, both the transmitter and the receiver have their radio modules turned on. In principle, this could be done by aligning all the nodes to agreed communication time slots, but the global synchronization tends to be counterproductive in terms of power consumption because of the independence of the actual communication needs and because of the additional overhead [9]. Two techniques can be adopted to cope with the lack of a global synchronization: *duty cycling*, in which the radio module of each node wakes up periodically and waits for the radio module of its peer to wake up before exchanging a packet, and *asynchronous wakeup*, in which the transmitter is assumed to be able to trigger the wakeup of the receiver.

Duty cycling solutions are further classified into *transmitter-initiated* schemes, where a preamble is sent by the transmitter to announce a packet, or *receiver-initiated* schemes, where a beacon is sent by the receiver to announce that it is ready to receive a packet. Wake-on radios are also available that perform duty cycling without involving the MCU [14]. Regardless of the flavour adopted, duty cycling imposes a trade off between power and performance. In fact, longer wakeup periods reduce the average power consumption at the cost of increasing the average latency. Moreover, the initiator of the communication has to send the preamble/beacon until the other party wakes up. Both the average and the worst cases depend on the length of the wakeup period. For instance, Contiki MAC implements the transmitter-initiated duty cycling scheme of the IEEE 802.15.4 protocol by using directly the data packets in place of the beacons. This implies that each packet is retransmitted until an ack is received or the wakeup period elapses.

Asynchronous schemes provide an attractive alternative my making use of wakeup receivers that can be either *in-band* or *out-of-band*, depending on the channel used for triggering the wakeup [3, 6, 5, 15, 4]. These solutions tend to be more efficient than duty cycling both in terms of communication cost (in that they avoid retransmissions) and in terms of latency (since the wakeup time is usually much shorter than the duty cycling period), but the listening power of the wakeup receiver adds to the average power consumption of the LPM.

Independently of the rendez-vous scheme adopted, packets exchanged among the nodes of dense WSNs have a non-negligible collision probability, that further impairs energy efficiency and performance by requiring re-transmissions. A *clear channel assessment* (CCA) process is used right after wakeup in contention-based protocols both by the transmitter and by the receiver to detect the activity on the channel. If no activity is detected at wakeup, the transmitter enters in transmit mode, while the receiver goes back to LPM. On the contrary, if some activity is detected at wakeup, the transmitter goes back to LPM and waits for the subsequent period to transmit its packet, while the receiver enters in receive mode and waits for the header of an incoming packet.

2.3.3 Overhearing

In dense WSNs each node receives and decode not only its own incoming packets, but also any other packet originated within the range if its antenna and possibly directed to other nodes. This phenomenon, known as *overhearing* of unintended packets, causes a sizeable waste of energy which is hard to be addressed by energy-aware algorithms. As a matter of fact, the energy required to receive an unintended packet is almost the same required to receive an intended one, while the percentage of unintended packets is much higher than 50% because of the density required to guarantee a redundant coverage of the target area and of the omnidirectional nature of the antennas usually installed on sensor nodes. In principle, three types of solutions can be adopted: i) using reservation-based protocols on top of duty cycling, which is usually not applicable in WSNs because of the lack of synchronization and of the variability of the topology, ii) out-of-band selective wakeup, which makes use of a separate signaling channel to wakeup the receiver while keeping all other nodes in a LPM, ad iii) unintended frame filtering, which exploits the hardware frame filtering capabilities provided by state-of-the-art radio transceivers [7] to turn off the radio module of the receiver as soon as the address of an unintended packet is received.

3. POWER STATE MODEL

This section presents a general power state model to be used to represent the DPM support provided by a given wireless sensor node and to evaluate to what extent it enables the exploitation of the idleness of the workload.

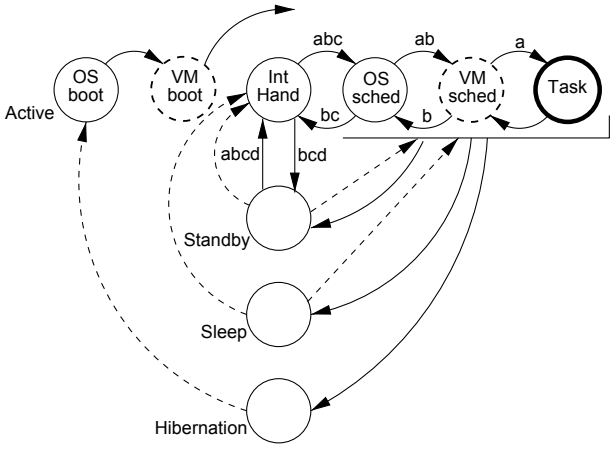

Figure 1: Power-state model including the transient states of the software stack.

Figure 1 shows the power-state diagram of a generic ultra-low-power MCU, as described in Section 2.1, providing one active mode and three LPMs. The active mode is split into several states to represent the macro-steps required at wakeup to resume the execution of a task running on top of the

operating OS and, possibly, of a virtual runtime environment, as described in Section 2.2. Dashed circles are used to denote optional states, which are to be considered only in case of virtualization. Solid arcs represent self-events, while dashed ones represent external interrupts.

According to the definition of the low-power states, self wake-up is enabled only from Standby, while external interrupts are required to wakeup from Sleep and Hibernation. It is also worth noticing that a complete boot is required when exiting from Hibernation, while data retention allows execution to resume directly when exiting from Standby and Sleep modes. Transitions represented on the right-hand side of the graph denote the possibility of entering low-power states directly from a code segment and resuming execution from the same point at wake-up at any level of the software stack.

According to the behavior described in Section 2.2, Contiki exploits the Standby state whenever all its running processes are waiting for scheduled timers or external events, but in order to keep control of the elapsed time it sets a periodic timer interrupt which wakes up the CPU every 10ms. Upon wake-up the interrupt handler evaluates if there are running processes which need to resume execution. If this is the case the control is passed to the scheduler, otherwise the CPU is turned off again soon. As mentioned in Section 2.2, the Darjeeling VM running on Contiki is a process which needs to resume at each timer interrupt in order to check for the status of its threads. If there are no threads ready to resume, the process is suspended until next timer interrupt. Labels a, b, c, and d in Figure 1 denote the transitions taken upon a timer interrupt in case of: a) a task ready to resume execution, b) a virtual runtime environment without tasks to resume, c) an OS with no processes to resume, and d) a wakeup directly filtered by the interrupt handler.

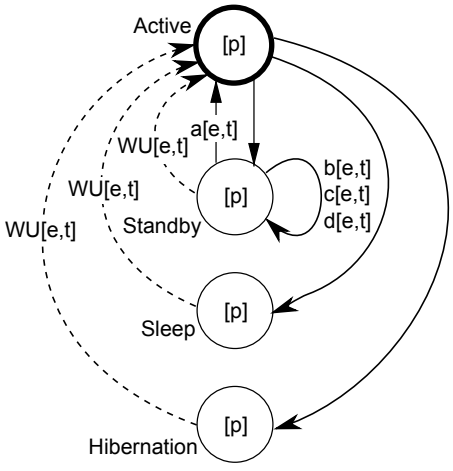

Figure 2: Power-state model with hidden transient states and self-loops.

Figure 2 provides a simplified version of the power state diagram where cases b, c, and d are represented as self-loops of the Standby state and annotated with the energy and time they require to be traversed. Similarly, the software overhead (both in terms of energy and time) is implicitly associated with the wake-up transitions from LPMs rather than being explicitly represented by the corresponding transient states of Figure 1. Each state of the diagram of Figure 2 is assumed to be annotated with its average power consumption. In order to adapt the model to MCU families providing more than one inactive state for each LPM, the number of power states can be incremented accordingly.

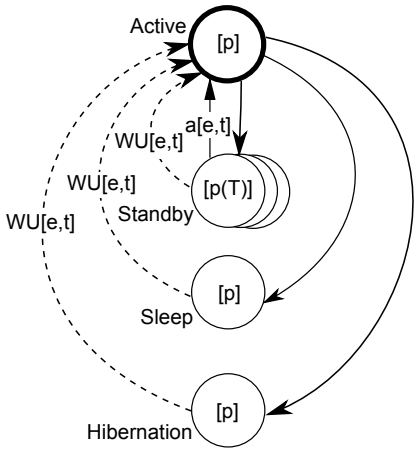

Figure 3: Power-state model with implicit self-loops.

A further abstraction is provided in Figure 3, where all the self loops that are traversed periodically are not represented any more because their overhead is directly accounted for in the average power consumption of the Standby state. Since such an overhead depends on the period of the timer interrupt (T), the Standby mode is represented as a family of parametric power states the average power consumption of which depends on the value of T. A similar approach can be used, in general, to incorporate into the average power consumption of a given state any self-loop (possibly going through multiple transient states) periodically traversed from that state.

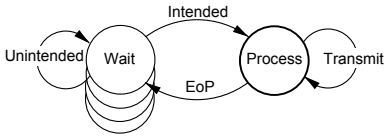

Figure 4: Functional state diagram of a wireless sensor node.

Figure 4 shows the top-level functional state diagram of a mote which exhibits two macro states: *Wait*, in which the node is waiting for incoming messages or events, and *Process*, in which the node executes a task. Wait is represented as a family of states corresponding to the different LPMs made available by the platform adopted, taking into account the MCU (as modelled in Figure 3), the support provided by the radio transceiver, and the communication solutions adopted (as discussed in Section 2.3). Transitions from Wait to Process are triggered either by incoming intended packets or by relevant events, while transitions from Process to Wait are triggered by *end-of-processing* (EoP) events. The reception of an unintended packet and the transmission of a packet are represented as self-loops of Wait and Process states, respectively, in that they do not cause any state transition.

For a given mote, the functional diagram has to be characterized in terms of average power consumption of the Process state and of all the Wait states available, and in terms of the time and energy required to traverse each edge. The annotated diagram can then be used to evaluate the capability of the mote to exploit the idleness of its workload, as exemplified in the following Section.

4. CASE STUDY

The models derived so far are used in this section to evaluate the energy efficiency of VirtualSense, an open-hardware ultra-low-power mote used here as a case study [10]. VirtualSense makes use of a TI MPS430F5418A MCU [8] to run a customized version of the Darjeeling VM [1] on top of Contiki OS [2], thus providing a Java-compatible virtual runtime environment. A TI CC2520 RF transceiver [7] is used for IEEE 802.15.4 wireless communication. In addition, VirtualSense mounts external real-time clock and flash memory to provide self-wakeup capabilities, accurate time measures, and data retention in all the LPMs of the MCU.

State name	**Power [μW]**	**WUt [ms]**	**WUe [μJ]**
Active	13440	n.a.	n.a.
Standby(T)	$14.67 + 0.30/T$	23.41	312.72
Sleep	$1.32 + 0.30$	23.41	312.72
Hibernation	$0.36 + 0.30$	560	4709.70

Table 1: Characterization of the power-state model of a VirtualSense mote with the MCU powered at 3V and clocked at 16MHz with a 4Kbyte VM heap.

Table 1 shows the parameters of the power-state model of Figure 3 characterized by means of real-world measurements performed on VirtualSense powered at 3V and clocked at 16MHz. Power values include the consumption of all the components, while wakeup time and energy from Hibernation include the cost of restoring the heap of the VM from the external flash memory.

The first row of Table 2, labelled *baseline*, reports the corresponding parameters of the functional state diagram of Figure 4, obtained with the default settings of VirtualSense, which exploits only the Standby LPM of the MCU (with a $T = 100$ms) and the LPM1 of the radio transceiver.

Thanks to the open hardware nature of the VirtualSense project and to the modular design of the mote [11], the following solutions have been implemented and tested in order to improve the exploitability of the idle periods of the workload, according to the discussion conducted in Section 2: i) the software stack has been modified in order to allow the exploitation of all the LPMs of the MCU [10], ii) the device driver of the radio transceiver has been modified in order to exploit the LPM2 [10], iii) the hardware frame filtering (FF) capabilities of the radio transceiver have been exploited to turn off the radio module as soon as the address of an unintended packet is received, iv) an ultrasonic (US) module has been developed to provide out-of-band asynchronous wakeup capabilities, and v) a simple out-of-band addressing scheme (USa) has been implemented to provide selective wake-up capabilities.

The impact of the advanced solutions on the parameters of the state diagram is shown in Table 2 for the most relevant configurations. The labels in each row report the name of the LPM of choice possibly followed by a suffix denoting the adoption of frame filtering (FF) or ultrasonic out-of-band wakeup solutions (US or USa). Moreover, prefix ML is used to denote the case of a memory-less application which does not require the external flash memory to restore the heap of the virtual machine. It is worth noticing that the solutions adopted cause a reduction of the waiting power of three orders of magnitude, without adding to much to the transition costs.

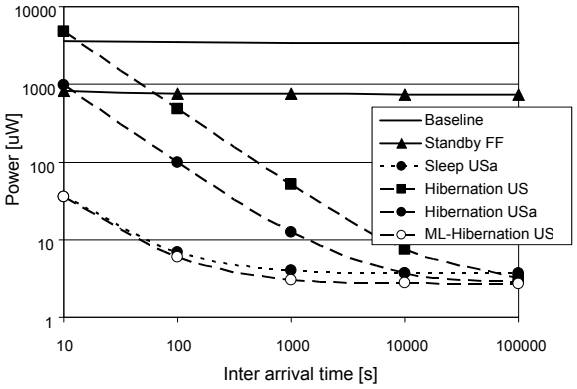

Figure 5: Average power consumption of a mote as a function of the inter-arrival time between incoming packets, for different low-power modes. Data refer to the case in which 20% of the packets received are intended packets.

Table 2 allows application developers to perform accurate power simulations to evaluate the exploitability of the idleness of the workload and to choose the best configuration of the motes adopted. For instance, Figure 5 shows the average power consumption of a generic VirtualSense node as a function of the inter-arrival time among the incoming packets, assuming that only 20% of them are intended packets. The curves are associated with the configurations of Table 2 to provide a direct comparison of their energy efficiency in different operating conditions. The choice of the most suitable configuration can reduce the average power consumption of up to four orders of magnitude with respect to the default settings of an ultra-low-power mote in case of long inter-arrival times.

5. CONCLUSIONS

The possibility of a building energy-neutral WSNs strongly depends on the capability of its nodes to exploit the idle periods to recover the energy spent to execute the tasks assigned to them. In fact, the power consumption of an active node is much higher than the average power supply provided by energy harvesters compatible with the typical requirements of a WSNs.

This paper has pointed out the main issues that impair the effectiveness of DPM techniques, and it has provided an overview of the state-of-the-art solutions available to address such issues. Moreover, it has proposed a power state

Configuration	Wait [uW]	Unintended [uJ]	Receive [ms]	Receive [uJ]	Transmit [ms]	Transmit [uJ]
Baseline	3,451.5	1,502.3	73.4	1,559.44	50.0	3,697.11
Standby	747.5	1,502.3	73.4	1,559.44	50.0	3,697.11
Standby FF	747.5	521.4	73.4	1,559.44	50.0	3,697.11
Sleep	734.4	1,502.3	73.4	1,559.44	50.0	3,697.11
Sleep FF	734.7	521.4	73.4	1,559.44	50.0	3,697.11
Sleep US	3.678	1,503.2	28.2	1,560.31	65.9	3,746.4
Sleep USa	3.678	3.87	428.2	1,563.31	515.9	13,555.2
Hibernation US	2.718	5,987.5	564.8	6,044.6	65.9	3,746.4
Hibernation USa	2.718	3.87	964.8	6,047.6	515.9	13,555.2
ML-Hibernation US	2.718	1,547.0	31.49	1,604.1	65.9	3,746.4
ML-Hibernation USa	2.718	3.87	431.49	1,607.1	515.9	13,555.2

Table 2: Parameters of the state diagram of Figure 4 for all the relevant configurations of VirtualSense. All the parameters were measured on a mote powered at 3V, with the MCU clocked at 16MHz and the ultrasonic module powered at 2V.

model to be used to evaluate the energy efficiency of a wireless sensor node, and it has illustrated its application on a representative case study. The results obtained on the case study demonstrate that a careful exploitation of the idleness of the workload can reduce the average power consumption of a wireless sensor node of up to four orders of magnitude.

6. REFERENCES

[1] N. Brouwers, K. Langendoen, and P. Corke. Darjeeling, a feature-rich vm for the resource poor. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pages 169–182, New York, NY, USA, 2009. ACM.

[2] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, LCN '04, pages 455–462, Washington, DC, USA, 2004. IEEE Computer Society.

[3] G. U. Gamm, M. Sippel, M. Kostic, and L. M. Reindl. Low power wake-up receiver for wireless sensor nodes. In *Proceedings of the 6th International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2010*, ISSNIP '10, pages 121–126, 2010.

[4] C. Hambeck, S. Mahlknecht, and T. Herndl. A 2.4Âţw wake-up receiver for wireless sensor nodes with -71dbm sensitivity. In *International Symposium on Circuits and Systems (ISCAS 2011)*, pages 534–537, 2011.

[5] X. Huang, P. Harpe, G. Dolmans, and H. de Groot. A 915mhz ultra-low power wake-up receiver with scalable performance and power consumption. In *Proceedings of the 37th European Solid-State Circuits Conference, ESSCIRC 2011*, pages 543–546, 2011.

[6] X. Huang, P. Harpe, W. Xiaoyan, G. Dolmans, and H. de Groot. A 915mhz ultra-low power wake-up receiver with scalable performance and power consumption. In *Proceedings of the 37th European Solid-State Circuits Conference, ESSCIRC 2011*, pages 543–546, 2011.

[7] T. Instruments. Cc2520 datasheet, 2013.

[8] T. Instruments. Msp430 ultra-low-power microcontrollers datasheet, 2013.

[9] V. Jelicic, M. Magno, D. Brunelli, V. Bilas, and L. Benini. Analytic comparison of wake-up receivers for wsns and benefits over the wake-on radio scheme. In *Proceedings of the 7th ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, PM2HW2N '12, pages 99–106, New York, NY, USA, 2012. ACM.

[10] E. Lattanzi and A. Bogliolo. A java library for event-driven communication in power-manageable reactive sensor nodes. In *Proceedings of NetWare*, pages 112–118. IARIA, 2012.

[11] E. Lattanzi and A. Bogliolo. Virtualsense: A java-based open platform for ultra-low-power wireless sensor nodes. *International Journal of Distributed Sensor Networks*, 2012, 2012.

[12] P. Levis and D. Culler. Maté: a tiny virtual machine for sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(5):85–95, Oct. 2002.

[13] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. Tinyos: An operating system for sensor networks. In *Ambient Intelligence*. Springer Verlag, 2004.

[14] G. Lu, D. D. Mingsen, and X. Song. Telosw: Enabling ultra low-power wake-on sensor networks. In *7th International Conference on Networked Sensing Systems (INSS)*, pages 15–18, 2010.

[15] S. J. Marinkovic and E. M. Popovici. Nano-power wireless wake-up receiver with serial peripheral interface. *IEEE Journal on Selected Areas in Communications*, 29(8):1641–1647, 2011.

[16] S. Microelectronics. Stm32l1 series, 2013.

[17] R. Müller, G. Alonso, and D. Kossmann. A virtual machine for sensor networks. *SIGOPS Oper. Syst. Rev.*, 41(3):145–158, Mar. 2007.

[18] M. Sha, G. Hackmann, and C. Lu. Energy-efficient low power listening for wireless sensor networks in noisy environments. In *Proceedings of the 12th international conference on Information processing in sensor networks*, IPSN '13, pages 277–288, New York, NY, USA, 2013. ACM.