Network-Adaptive Management of Computation Energy in Wireless Sensor Networks

Fabrizio Mulas
University of Cagliari
Cagliari, Italy
fabrizio.mulas@sc.unica.it

Andrea Acquaviva
Politecnico di Torino
Torino, Italy
andrea.acquaviva@polito.it

Salvatore Carta
University of Cagliari
Cagliari, Italy
salvatore@unica.it

Gianni Fenu
University of Cagliari
Cagliari, Italy
fenu@unica.it

Davide Quaglia
University of Verona
Verona, Italy
davide.quaglia@univr.it

Franco Fummi
University of Verona
Verona, Italy
franco.fummi@univr.it

ABSTRACT

Today's sensor nodes can be equipped with powerful microcontrollers to address the increasing need of real-time processing of sensed data. For instance, body sensor networks for gesture recognition require filtering of acceleration values at line rate. This requirement imposes a paradigm shift with regard to more traditional sensor networks characterized by low activity duty cycles. Therefore, energy conservation strategies applied to wireless sensor nodes to increase their lifetime must take into account computation power rather than focusing only on communication power. In this paper we present a novel approach which aims at exploiting the knowledge of network status to optimize the power consumption of the node microcontroller. The proposed approach is tested in various network conditions, both synthetic and realistic, in the context of IEEE 802.15.4 standard. Experimental results demonstrate that the proposed approach allows to achieve power savings of up to 70% with minimum performance penalty.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications, Distributed databases*

General Terms

Design, management

Keywords

Wireless sensor networks, low-power

1. INTRODUCTION

Wireless sensor networks (WSN) have recently gained more and more attention in human computer interaction (HCI) and e-health applications for gesture recognition and body posture monitoring. In these applications, sensor nodes elaborate data from body mounted accelerometers or gyroscopes to reconstruct movements. Hence, node's microcontroller must perform integrations or trigonometric function computations in real-time. As a consequence, the power consumption of the processing components of the node is much higher (i.e. imposing a duty cycle of 50% or more) than in traditional WSN applications such as infrastructure monitoring or video surveillance where activity duty cycles of nodes are on the order of 1%.

As a consequence, a paradigm shift in the design of energy management strategies for wireless sensor networks is required, where management of microcontroller's energy plays a central role together with communication energy reduction.

Figure 1 depicts a typical data flow in a sensor network application. The producer node acquires data by using its sensors, then it processes data by using a microcontroller and puts results in the transmission queue to be sent over the network. These operations must be performed at the proper speed to match application performance requirements. Traditionally, in wireless transmissions, packet reception is confirmed by sending back an acknowledgment. If the quality of the radio link is poor or the receiver cannot receive data then no ack comes back and the packet is kept in the queue to be retransmitted. Furthermore in contention-based access protocols (e.g., the IEEE 802.15.4 [12]) the packet transmission is delayed if the channel is busy. As a consequence, at the producer side the transmission queue may become full thus wasting CPU power. Indeed, the data processed by the microcontroller will be either discarded before entering the queue (drop tail policy) or will cause the dropping of packets in the head of the queue (drop head policy). Independently from the dropping policy (for which we do not make any assumption here), since typically the size of the transmission queue is small (16 to 32 packets), most of the data processed by the microcontroller will be lost in case of relatively long network congestion conditions. This means that in such cases it is more convenient to slow-down processing speed and save microcontroller's power. We consider long periods as those leading to transmission queue overflow.

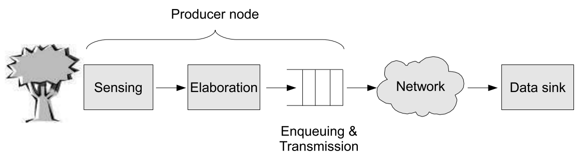

Figure 1: Data flow in a sensor network application

This means that congestion periods smaller than the time needed to fill in the transmission queue do not cause the activation of our speed slow-down policy.

The efforts done in the past to decrease energy spent in transmission/reception and the increasing role of computation in today's applications have made CPU power consumption more critical for node's power budget. It has been shown that the CPU is one of the most consuming components of a modern wireless sensor node ([10], [14]). In particular [14] performed a very accurate profiling of energy consumption of the widely adopted Mica2 sensor node. Their results show that the CPU power consumption ranges from 28% to 86% of the total power consumed and roughly 50% on average. Moreover, depending on the node application domain, further activity can be assigned to the CPU (i.e. data filtering), thus raising its power consumption share.

Several energy management approaches aiming at optimizing network lifetime have been proposed in the past (see [4] for an overview); most of them focus on communication power reduction [16, 20]. Concerning processing power, recent works exploit topology and communication range optimization to shutdown the cores [11, 13, 19]. A distributed power management approach is presented in [21] where coordination among nodes aims at optimizing the timeout of their sleeping periods. Route and network activity information is exploited in [18] to decide when to turn the node into the sleeping status. Other works investigated Dynamic Frequency Scaling (DFS) and Dynamic Voltage Scaling (DVS) [15] to reduce power consumption when the CPU is in idle state.

The limit of those approaches is that they do not exploit the knowledge of network conditions to save power. In this paper we present a novel approach where network information is exploited to independently adapt the processing rate of each sensor node to the network conditions. Since WSN protocol stacks lack of rate-control algorithms such as those implemented in TCP, the occupation of MAC queue has a linear relationship with the difference between application production rate and network consumption rate. For this reason, we implemented our network-adaptive approach directly at MAC level. Nevertheless, we believe the proposed policy could be applied also in presence of rate-control protocols by appropriate tuning of the control law.

To adapt the processing rate to the network condition a control strategy has been designed and implemented, that must satisfy conflicting objectives. From one side, it must be reactive enough to detect network congestion conditions early enough to: i) allow power saving when the queue is filling up, and ii) to avoid queue depletion that would compromise quality of service. On the other side, large reactivity will lead to frequent speed switchings, that are not desirable because of their time overhead (which is taken into account in our experiments). We studied a non-linear control approach that is able to address these objectives. We implemented it on a power-aware wireless sensor network simulator that models a network of TI CC2430 nodes, each consisting of a processing core and a radio interface running the 802.15.4 MAC layer. To evaluate the effectiveness of the proposed strategy, we performed experiments using a realistic human body monitoring application case study to detect real network congestion conditions.

The paper is organized as follows: Section 2 explains the energy management strategy, Section 3 describes the target platform and modeling approach while Section 4 describes the experimental results. Section 5 concludes the paper.

2. COMPUTATION ENERGY MANAGEMENT

Modern microcontrollers support various operating clock frequencies that can be programmed by the software applications through a dedicated API. Typically, these frequency values are obtained by dividing the maximum frequency by a factor of two. Indeed, frequency scaling is obtained by a pre-scaler hardware module acting on the clock signal entering the microcontroller core. The clock scaling can be exploited by an energy management policy to save power when the production of new packets would be useless since packet transmission slows down and the transmission queue is getting full. This fact happens when the channel is busy, the quality of the radio link is poor (e.g., due to interference, path loss, obstacles) or the receiver is sleeping or overloaded. In this case, the processing speed can be decreased until packet transmission rate increases again. Therefore, the transmission queue can be used to monitor the network status.

When the network quality is low, the transmission queue of a node starts filling-up at a speed depending on the producer rate (the microcontroller) and, if the condition persists, it can become full. Thus, to save power on the microcontroller, a simple approach could be to switch to the lowest possible frequency value (or even shut-off) when the transmission queue becomes full and restore the maximum frequency when the queue starts to be depleted again. We called this simple algorithm *On-Off* (OO) and let *set-point* be the level of the queue which triggers frequency change. A more aggressive approach would be to reduce the processor speed before the queue is full (i.e., using a lower set-point) to save more power. However, there are two side effects from the performance viewpoint. First, the average queue level is lower, which implies that there could be less packets to be transmitted once the congestion period finishes. That could worsen the quality of service. Second, if the algorithm is not reactive enough, the processor speed at the end of the congestion period could be lower than the maximum, thus leading to a lower transmission rate on average (again a QoS worsening). As such, there is a trade-off between power saving and performance that should be explored during the design of the control rule.

To implement a more aggressive approach with a limited impact on performance, we explored the use of a non-linear feedback control which has been developed in the field of multiprocessor systems to regulate the speed of a pipeline of cores [3].

2.1 Non-linear Feedback Control

In this section we describe our proposed non-linear speed

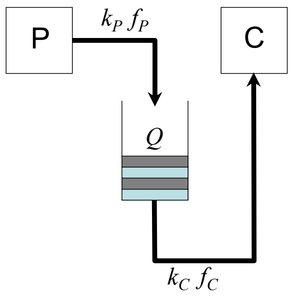

Figure 2: Producer (P) / Consumer (C) architecture

scaling technique, namely *HQ*. Let us derive a dynamical model of the two-stage producer-consumer architecture represented in Figure 2. In our system the microcontroller (producer) inside the node produces data that are injected in the transmission queue of the node network interface (consumer). Let Q be the occupancy of the transmission queue (by definition, Q is an integer non-negative number) and f_P be the producer clock frequency. The network consumption rate f_C is an external constraint. This rate is time-varying and depends on network conditions.

Denote as $k_P f_P$ the data rate of the producer processor, and let $k_C f_C$ be the data rate of the consumer, with k_P and k_C being proper positive gains. To facilitate system modeling and controller design, we define $\overline{Q}(t)$ as a real-valued (i.e., "fluid") approximation of Q, and we consider the following dynamical model:

$$\dot{\overline{Q}}(t) = k_P f_P(t) - k_C f_C(t) \tag{1}$$

where $\overline{Q}$ plays the role of the system output to control, f_P is the user-adjustable control input and f_C represents an external disturbance term.

The frequency f_P can take on values over a discrete set. Let Q_{cap} be the queue capacity and let $Q^* = Q_{cap}/2$ be a convenient set-point for the queue occupancy.

Denote as follows the "error variable" e to be regulated.

$$e = Q - Q^* \tag{2}$$

The control algorithm 1 processes the current queue occupancy error $e[k]$ and its previously observed value $e[k-1]$.

Besides, we introduce a *dead zone* centered around the set-point where the algorithm does not change the frequency and we denote it as 2Δ (by definition Δ is a non-negative integer). The dead zone has the purpose to limit the frequency changes.

Algorithm 1 HQ controller

```
Every trigger instant do:
$computeFrequency(queuelevel)$
1: if $e[k] < -\Delta \quad AND \quad e[k] \leq e[k-1]$ then
2:     $increaseProducerFrequency()$
3: else if $e[k] > \Delta \quad AND \quad e[k] \geq e[k-1]$ then
4:     $decreaseProducerFrequency()$
5: end if
```

In our sensor node the microcontroller can have three different frequencies (see Table 1), so three steps are possible. The following functions change the frequency:

increaseProducerFrequency(): it raises the frequency up to an higher step, if not already at maximum.

decreaseProducerfrequency(): it scales the frequency down to a lower step, if not already at minimum.

3. TARGET PLATFORM AND SIMULATION MODEL

3.1 Simulation Framework

The efficient simulation of wireless sensor nodes requires the capability of modeling both their behavior/architecture (HW and SW) and the complex communication environment in which they operate (the network). HW/SW co-simulation follows the scheme proposed in [7] and targets a generic architectural template in which software (that will eventually run without changes on the actual board) accesses one or more hardware devices that have to be designed. This scenario maps onto a so-called ISS-centric co-simulation model consisting of an instruction set simulator (ISS) running the application and the operating system that interfaces through its drivers to the hardware models specified by a hardware description language such as SystemC [1]. The interaction between simulated software and hardware modules is simplified by the fact that many embedded platforms support memory-mapped HW access, i.e., CPU accesses external registers through memory read/write operations. Therefore, the ISS is modified to redirect read/write operations for specific addresses to the hardware simulation kernel which updates the status of the corresponding HW modules.

The simulation infrastructure enables accurate power estimation thanks to the following main features: i) timing synchronization between SystemC model and ISS; ii) power annotation of CPU states and hardware components [8]. To achieve effective evaluation of power management strategies, power model of HW components has to support voltage and clock frequency scaling as well as shutdown states. To this purpose, each hardware component will be associated with a power state machine [2] and a power consumption value will be associated to each state. Parametric states will be also used for specific components for which the power consumption depends on the voltage and clock frequency, such as the microcontroller. The transitions among power states of SystemC modules will be controlled by the software running on the ISS through the dedicated power information protocol as described in [8].

3.2 Network Model

To effectively simulate the wireless network to which sensor nodes are connected, a SystemC collection of components has been created to reproduce packet transfer over a radio channel [9]. In this way the same tool used to simulate part of sensor nodes is seamlessly exploited to reproduce network behavior, thus gaining simulation efficiency. Figure 3 shows the architecture of the SystemC Network Simulation Library. White boxes represent SystemC modules while gray boxes are pure C++ classes; black arrows represent connections through SystemC ports; bold black arrows represent inheritance and round edges denote relationships through object references.

Module `Node_t` models a generic network node; it has three input ports (generic input, network input, received signal energy input) and an output port (network output).

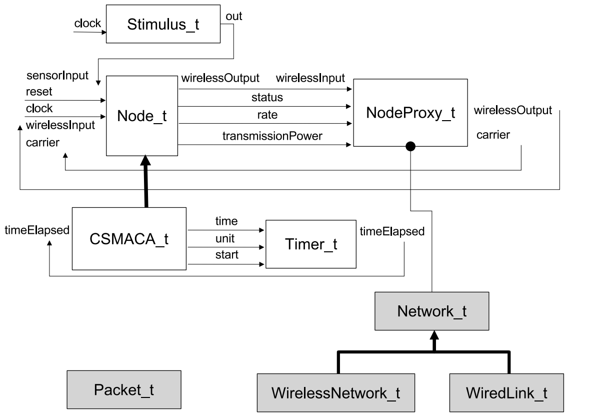

Figure 3: Architecture of the SystemC Network Simulation Library

Module Node_t has a set of properties which are used by the simulation framework to reproduce network behavior. Nodes have a *state* which can be running, off, or sleeping to save power. Transmission *rate* represents the number of bits per unit of time which the interface can handle; it is used to compute the transmission delay and the network load. The *transmission power* is used to evaluate the transmission range and the signal-to-noise ratio. Node state, transmission rate and transmission power can be changed during simulation to accurately simulate and evaluate power saving algorithms. As shown in Figure 3, sub-classes of Node_t can be created to describe actual nodes. The sub-class specifies the functional and timing behavior of the node (e.g., the CSMA/CA policy); it can define additional ports and whether data sampling is triggered by a change of the value on the data input port or scheduled by the node itself. Instances of the module Timer_t can be connected to user-defined nodes to implement timed actions.

The data input port of each node can be bound to an instance of the module Stimulus_t which reproduces a generic environmental data source. Class Network_t is the core of the network simulator. It reproduces the behavior of the channel and manages the packet forwarding from the source node to the destination nodes: transmission delay, path loss, collisions, and the state of destination nodes are taken into account. Module NodeProxy is the interface between nodes and the network and each instance of Node must be bound to a different instance of NodeProxy. Each node interacts with its own nodeproxy by using SystemC signals only, while nodeproxies interact with Network_t through object references. By using NodeProxy, nodes can be designed as pure SystemC modules without object references to other non-SystemC classes; this approach enables the use of traditional hardware verification and synthesis tools. Exchanged packets are modelled by the Packet_t class which contains the address of source and destination nodes, the packet length and a general-purpose payload field. Packet definition can be changed since its fields are used by model-specific code while only its size is used by the network; general purpose classes are independent from the packet structure since it is specified through the template mechanism.

3.3 Target Platform Model

This section describes the modeling of a wireless sensor node, called AquisGrain-2 provided by Philips [6] and designed for body-worn smart medical sensors. The main modules belonging to a sensor node based on the AquisGrain-2 platform are the following: (I) ZigBee SW stack; (II) Intel 8051 CPU, memory (ROM, RAM, flash) and I/O ports; (III) IEEE 802.15.4 RF transceiver; (IV) Sensors/actuators. Figure 4 shows how the model of the AquisGrain-2 is mapped onto the co-simulation framework. The ISS used to model the Intel 8051 CPU is uCsim [5]. SystemC is used to model HW devices. The overall HW/SW configuration consists of the following entities:

- SystemC RTL model of the accelerometer chip;
- SystemC RTL model of the UART/SPI device, which is attached to an input/output port of the accelerometer to exchange data with the CPU;
- C application retrieving data coming from the accelerometer and sending them over the Network; this application is executed by using uCsim;
- SystemC model of the Network implemented through SCNSL.

4. EXPERIMENTAL RESULTS

In this section we assess the effectiveness of the two feedback control power management policies described in Section 2, namely OO and HQ. We also compare them against the case in which there is not power management (we called this last case *NO*); we evaluate the trade-off between obtained power savings and quality of service impact. The policies are implemented within the network interface driver of the sensor node, so that their cost in terms of performance and power is taken into account. The benchmark consists in sampling acceleration values, performing a simple filtering operation and sending them over the network. The acceleration sample rate depends on the speed of the microcontroller, as shown in Table 1 where the power consumption of the core at the various frequencies is reported too. It is worth noting that in this benchmark, when the network is not congested, the bottleneck is represented by the processing rate rather than the network bandwidth. This is a typical situation for sensor network applications. As such, even at maximum speed, the transmission queue depletes. Therefore, the feedback control strategy exploits the channel congestion periods, that cause the saturation of the queue,

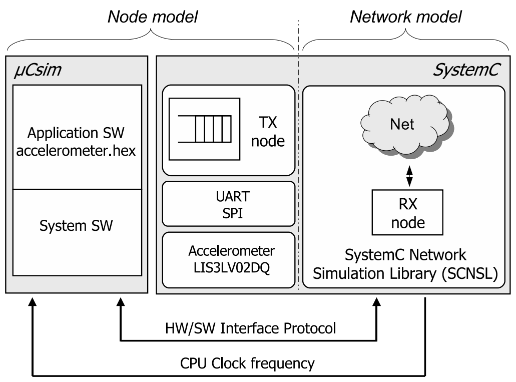

Figure 4: Co-simulation of the AquisGrain-2 node and of the wireless network

Table 1: Power vs performance characteristics of the microcontroller

Frequency	*Output Rate*	*Power*
8 MHz	2 KHz	8.25 mW
16 MHz	4 KHz	14.85 mW
32 MHz	8 KHz	31.35 mW

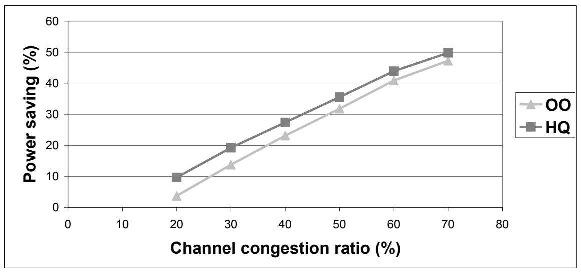

Figure 5: Power saving (%) as a function of the channel congestion ratio

to save the power spent by the microcontroller. To show the dependency of the proposed approach on network conditions, we modeled different network congestion patterns, both synthetic and realistic. We represent the congestion interval length as a fraction of the total transmission time. In Section 4.1 we consider a coarse grained pattern with sporadic events of variable length. This scenario emulates temporary disturbances such as a radio interference. Instead in Section 4.2 we consider a fine grained pattern, where congestion intervals alternate to free intervals, shaping a periodic square wave with a duty-cycle of 50%. In these experiments we vary the length of the wave period. Finally, in Section 4.4 we model a real case scenario.

In all the experiments the transmission buffer length of the network interface is set to 15 packets and the set-point (expressed as a fraction of the buffer length) is fixed at 1/2 for the HQ and 14/15 for the OO. Each simulation ends after the transmission of a given amount of packets over the network. In these experiments we focus on the microcontroller power. As such, power consumption of the radio is not included. Note that there is a negligible delay for changing the speed at run time being this obtained by programming a pre-scaler placed on the clock path to the core.

To determine the impact on performance of the power management actions we consider the effective transmission rate, i.e., the number of packets sent per time unit within time intervals free of congestion. Performance impact must be compared, for the same channel conditions, against the NO case (i.e., not power management), taking into account the obtained power savings.

4.1 Coarse Grained Channel Congestion

The Figure 5 reports the power saving (with respect to NO) of one sensor node over the channel congestion ratio (that is the congestion interval length as a fraction of the total transmission time). It clearly shows that power saving is proportional to the channel congestion ratio.

This is because longer the channel is busy, longer the frequency is held down. It is worth noting that we reached 50% of power saving in our best setup, but it can reach higher values with higher channel congestion ratio, up to the theoretical power saving upper limit of about 73%, obtained by

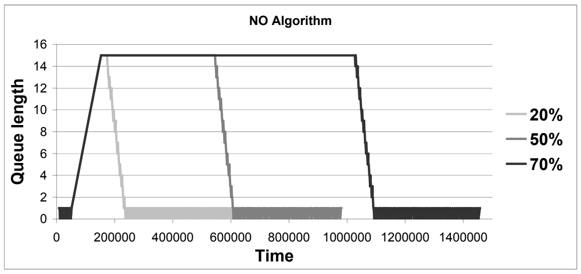

Figure 6: Queue length over time without frequency scaling

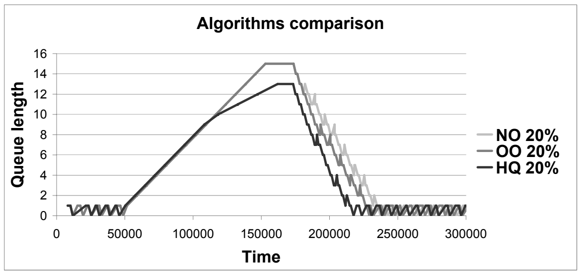

Figure 7: Comparison of queue occupancy as a function of time and with different power management strategies

always keeping the frequency at its lower level. To better understand these results, we can analyze the queue behaviour; Figure 6 shows the queue length over time with different percentages of channel congestion and without frequency scaling. It is possible to see that bigger the congestion ratio, longer the queue stays full; if frequency scaling were active, bigger the congestion ratio and longer the frequency would be held down.

We experimentally observed that for each case (NO, HQ, OO) the effective transmission rate is constant as a function of the channel congestion ratio, but it has different values for each of them. With respect to NO, OO impacts the performance of 2.6% and HQ of 6.5%. The HQ, being the more aggressive, shows the worst performance value. On the other side, this is compensated by the fact that it has the best power saving result (Figure 5). Conversely, being more conservative, the OO algorithm imposes a smaller impact on performance but it is less effective from a power saving viewpoint.

Later in this section we will explore the trade-offs available by configuring the parameters of these algorithms.

Figure 7 compares the level of the queue over time for the three algorithms for a given percentage of channel congestion (20%). It is evident that power management policies deplete the transmission queue due to the frequency scaling on the microcontroller. Once the queue is empty, the transmitter has to wait leading to a slow-down of the transmission rate (QoS worsening).

4.2 Fine Grained Channel Congestion

In this section, the chosen network conditions cause frequent changes in the queue level, as shown in the two plots in Figure 10, thus offering a more dynamic behavior to test the feedback policies. The effective transmission rate for

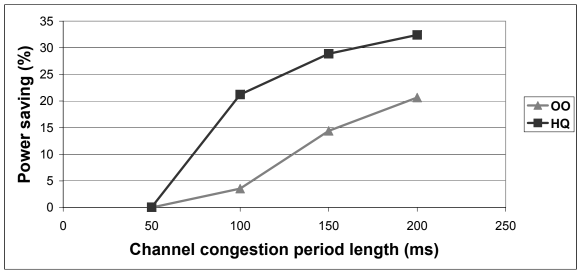

Figure 8: Power saving as a function of the channel congestion period length

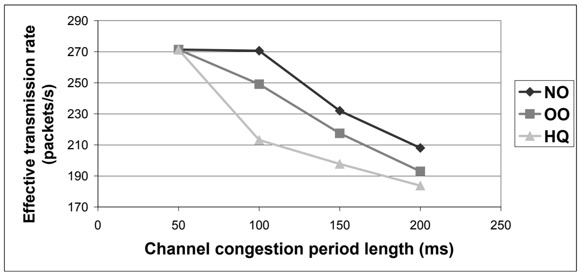

Figure 9: Effective transmission rate as a function of the channel congestion period length

each algorithm is not constant anymore, but it decreases as the congestion period length increases as shown in Figure 9. As with the previous case, the HQ saves more power than OO (Figure 8), even if it has a slightly higher performance impact (Figure 9).

It can be observed that at 50 ms of congestion period the effective transmission rate is the same for all the algorithms. In this case the queue always stays under the set-point (see Figure 10a), i.e., frequency scaling is never triggered.

The results in Figure 9 show that the effective transmission rate depends on the congestion period length. This can be justified by analysing the queue occupancy over time depicted in the two plots in Figure 10. For the sake of clarity, here we show only the case in which no power management is applied for different values of the period length (the behaviour of the other algorithms is similar). Both plots refer to the transmission of the same number of packets. It is worth noting that the falling edge of the peaks is that of maximum transmission speed, because packets accumulated in the buffer are consumed at the network rate after the congestion, being not limited by the CPU speed. When congestion periods are small such that the queue never becomes full, the effective transmission rate is independent from the congestion period as expected because the sum of intervals in which the queue is empty is constant. This is depicted in Figure 10a and it corresponds to congestion periods of 50 ms and 100 ms. On the other side, if the buffer saturates as shown in Figure 10b, the total time in which the queue is empty becomes larger because the buffer does not serve as packet reservoir. In practice, there are less opportunities to transmit at higher rate exploiting the full buffer situation. This situation holds also for congestion periods of 150 ms and 200 ms. From a power saving perspective, Figure 10b shows that by increasing the period length, the queue keeps saturated for a longer time, giving more opportunities to slow down the frequency. This is the reason why the power saving increases together with the congestion length.

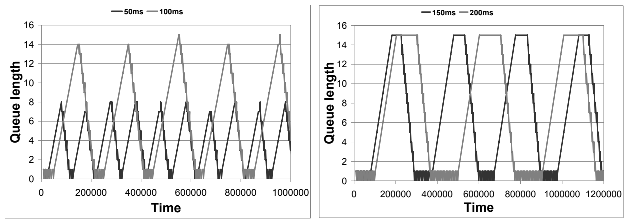

Figure 10: Queue length over time without frequency scaling for congestion length of: a) 50 ms and 100 ms; b) 150 ms and 200 ms

4.3 Parameters Tuning

By tuning some parameters of the algorithms it is possible to achieve a variety of power/performance trade-offs. In particular, by changing the set-point such that it corresponds to different fractions of the buffer length: 1/4, 1/2, 3/4 for the HQ and 1/2, 3/4, 14/15 for the OO. It must be noted that it is not possible to compare the set-points between the two algorithms, because they are used in a different way, as explained before. Hence, we chose the above set-points observing the average queue level over time, picking up the extreme values in the following manner:

- upper set-points (3/4 for HQ and 14/15 for OO): minimum value (respectively for each algorithm) that causes an average queue level equals to the NO case;
- lower set-points (1/4 for HQ and 1/2 for OO): maximum value (respectively for each algorithm) that causes an average queue level equals to the case where the CPU frequency is held at minimum level.

After that, we chose another value (for each algorithm) in the middle (1/2 for HQ and 3/4 for OO). Furthermore, we performed these experiments only for the fine grained case and at a given channel congestion period length (150 ms).

In order to drive practical rules for parameter selection, we plotted the Pareto optimal configurations in Figure 11, that highlights the achievable trade-offs between energy saving and performance impact. Depending on the application requirements, different configurations can be selected and therefore different parameter settings. For instance, if a more aggressive policy is needed at the price of a slightly higher performance penalty, the HQ algorithm can be set to 1/4, which provides the highest power saving and the worst effective transmission rate. Conversely, if the highest possible QoS is required, the OO algorithm should be used with a set-point of 14/15.

4.4 Realistic Case Study

In order to test our algorithms in real life situations, we simulated a wireless sensor network in a car manufacturing industry, used to control the activities of workers in the car assembly line. The sensors are placed on the body of workers to monitor their checking procedure (a full description of this scenario is described in [22] and [17]).

We statistically reproduced the same transmission patterns of that scenario in our simulator, using a two-state

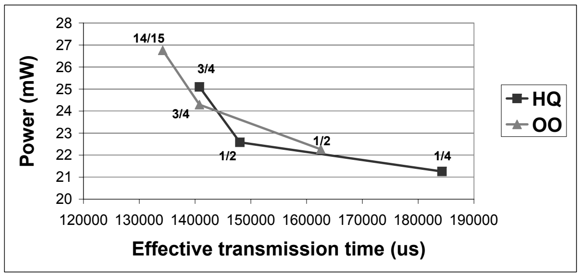

Figure 11: Pareto optimal configuration for HQ and OO algorithms

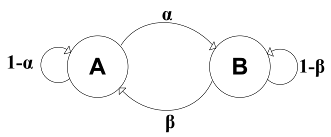

Figure 12: Two-state Markov chain

Markov chain to characterize the transmission activity of each node. Figure 12 depicts the Markov chain; in our case there are two states, named A and B, corresponding to a non-transmission and transmission state, respectively. The transition probability from the two states is regulated by α and β parameters obtained through several realistic data traces taken from the scenario described above. When the simulation begins, α and β parameter values are passed as parameters to each instantiated node, so that they generate a network traffic statistically similar to the realistic one. We performed several experiments by varying the number of nodes of the network, to simulate various channel congestion levels.

Figure 13 shows the results obtained on the realistic case study and confirms that through network-aware speed control algorithms it is possible to save a considerable amount of power (up to about 70%). In this case study we also found that performance impact is similar to that previously showed in coarse/fine grained cases.

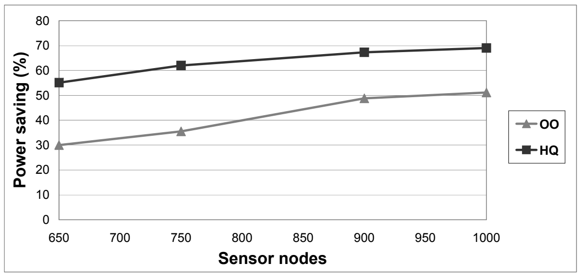

Figure 13: Power saving (%) as a function of the number of sensor nodes

5. CONCLUSIONS

In this paper we presented a feedback control strategy to adapt the computation power of the CPU core of a sensor node to the network conditions. The technique exploits periods of congestion to scale down processing speed depending on the occupancy of the transmission queue at the MAC level. The proposed approach has been validated against various network conditions, both synthetic and realistic. Experimental results performed on a realistic case study demonstrated that the non-linear feedback control law allows to achieve power savings of up to 70% with minimum performance penalty, depending on the congestion duration. We evaluated the proposed policy using a wireless sensor network simulator implementing IEEE 802.15.4 transmissions.

6. REFERENCES

[1] IEEE Std 1666 - 2005 IEEE Standard SystemC Language Reference Manual. *IEEE Std 1666-2005*, pages 1–423, 2006.

[2] L. Benini and G. de Micheli. System-level power optimization: techniques and tools. *ACM Trans. Des. Autom. Electron. Syst.*, 5(2):115–192, 2000.

[3] S. Carta, A. Alimonda, A. Pisano, A. Acquaviva, and L. Benini. A control theoretic approach to energy-efficient pipelined computation in mpsocs. *Trans. on Embedded Computing Sys.*, 6(4):27, 2007.

[4] X. Cui, X. Zhang, and Y. Shang. Energy-saving strategies of wireless sensor networks. In *IEEE 2007 International Symposium on Microwave, Antenna, Propagation, and EMC Technologies For Wireless Communications*, 2007.

[5] D. Drótos. μCSim: Software Simulator for Microcontrollers. *http://mazsola.iit.uni-miskolc.hu/˜drdani/embedded/s51/*.

[6] J. Espina, T. Falck, and O. Mülhens. *Network Topologies, Communication Protocols, and Standards.* In: Yang, G.Z. (ed): Body Sensor Networks, pp. 145-182, Springer, London, England, 2006.

[7] F. Fummi, G. Perbellini, M. Loghi, and M. Poncino. Iss-centric modular hw/sw co-simulation. In *GLSVLSI '06: Proceedings of the 16th ACM Great Lakes symposium on VLSI*, pages 31–36, New York, NY, USA, 2006. ACM Press.

[8] F. Fummi, G. Perbellini, D. Quaglia, and A. Acquaviva. Flexible energy-aware simulation of heterogenous wireless sensor networks. In *DATE*, pages 1638–1643, 2009.

[9] F. Fummi, D. Quaglia, and F. Stefanni. A SystemC-based framework for modeling and simulation of networked embedded systems. In *Proc. of ECSI Forum on Specification and Design Languages (FDL'08)*, pages 49–54, 2008.

[10] M. Hempstead, N. Tripathi, P. Mauro, G.-Y. Wei, and D. Brooks. An ultra low power system architecture for sensor network applications. *SIGARCH Comput. Archit. News*, 33(2):208–219, 2005.

[11] H.-C. Jang and H.-C. Lee. Efficient energy management to prolong wireless sensor network lifetime. In *ICI*, 2007.

[12] LAN/MAN Standards Committee of the IEEE Computer Society. IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (LR-WPANs). Sept. 2006.

[13] C. Lin, Y.-X. He, and N. Xiong. An energy-efficient dynamic power management in wireless sensor networks. 2006.

[14] V. Shnayder, M. Hempstead, B. rong Chen, G. W. Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 188–200, New York, NY, USA, 2004. ACM Press.

[15] A. Sinha and A. Chandrakasan. Dynamic power management in wireless sensor networks. *IEEE Des. Test*, 18(2):62–74, 2001.

[16] I. Slama, B. Jouaber, and D. Zeghlache. Optimal power management scheme for heterogeneous wireless sensor networks: Lifetime maximization under qos and energy constraints. In *Third International Conference on Networking and Services (ICNS'07)*, 2007.

[17] T. Stiefmeier, D. Roggen, and G. Troster. Fusion of string-matched templates for continuous activity recognition. *Wearable Computers, 2007 11th IEEE International Symposium on*, pages 41–44, Oct. 2007.

[18] H. Wang, W. Wang, D. Peng, and H. Sharif. A route-oriented sleep approach in wireless sensor networks. In *CS*, 2007.

[19] X. Wang, J. Ma, and S. Wang. Collaborative deployment optimization and dynamic power management in wireless sensor networks. In *GCC*, 2007.

[20] A. S. Zahmati, N. M. Moghadam, and B. Abolhassani. Epmplcs: An efficient power management protocol with limited cluster size for wireless sensor networks. In *ICDCSW*, 2007.

[21] N. H. Zamora, J.-C. Kao, and R. Marculescu. Distributed power-management techniques for wireless network video systems. In *DATE*, 2007.

[22] P. Zappi, T. Stiefmeier, E. Farella, D. Roggen, L. Benini, and G. Troster. Activity recognition from on-body sensors by classifier fusion: sensor scalability and robustness. *Intelligent Sensors, Sensor Networks and Information, 2007. ISSNIP 2007. 3rd International Conference on*, pages 281–286, Dec. 2007.

Fabrizio Mulas received the B.S. degree in electronic engineering from the University of Cagliari, Cagliari, Italy, where he is currently working toward the Ph.D. degree in computer science.

In 2007-2008, he spent six months at the Integrated Systems Laboratory, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, as a Guest Researcher about conception and development of software algorithms and policies for dynamic resource management in multiprocessor systems. His scientific activities mostly concern soft real-time scheduling, power management in wireless sensor networks, Linux kernel activity monitoring, management techniques for addressing variability/reliability, and aging problems in next-generation hardware components.

Andrea Acquaviva received the Ph.D. degree in Electrical Engineering at Bologna University in 2003. Since 2008 he is with the Department of Computer Engineering and Automation at Politecnico di Torino (Italy). He was also visiting researcher at the Ecole Politechnique Federale de Lausanne (EPFL) from 2005 to 2007. Research interests of Prof. Andrea Acquaviva focus mainly on: (i) parallel computing for distributed embedded systems such as multicore and sensor networks; (ii) simulation and analysis of biological systems using parallel architectures. Researches by Prof. Andrea Acquaviva (between 2000 and 2008) yielded more than 60 papers in international journals and peer-reviewed international conference proceedings as well as 6 book chapters.

Salvatore Carta graduated (summa cum laude) in Electronic Engineering at the University of Cagliari, in 1997. He received a Ph.D. degree in Electronics and Computer Science from Cagliari University in 2003. In 2005, Salvatore Carta became an Assistant Professor in Computer Science at the University of Cagliari (Italy).

Research interests of Salvatore Carta focus mainly on architectures, software and tools for embedded and portable computing, with particular emphasis on: (i) Operating systems, middleware and software for multiprocessor-systems-on-chips; (ii) Networks-on-chip; (iii) reconfigurable computing. He is author of more than 20 papers in these fields in the last three years. In the last two years he is also working in the fields of recommendation systems and social networks. He is author of some papers in these fields.

Gianni Fenu is associate professor of computer science at the Department of Computer Science at University of Cagliari (Italy) and has research interests in the area of Computer Network and Cloud Computing. He's Head of the degree course of Computer Science, Coordinator of Development Projects and published about 40 scientific papers on international Journals and Conferences on the following main topics: Computer Network, High Speed Network, Wired Network and Information Systems Development.

Davide Quaglia received the Ph.D. in Communication Engineering in 2004 at Politecnico di Torino. Since 2005 he is Assistant Professor at the Computer Science Department of Università di Verona. His main research interests concern network modeling, co-simulation, middleware for networked embedded systems. He published more than 30 papers on these areas.

Franco Fummi received the Ph.D. in Electronic and Communication Engineering in 1994 at Politecnico di Milano. In 1996 he obtained the position of Assistant Professor in Computer Science at Politecnico di Milano. In July 1998 he obtained the position of Associate Professor in Computer Architecture at Università di Verona. Since March 2001 he is Full Professor in Computer Architecture at Università di Verona. His main research interests concern hardware description languages and electronic design automation methodologies for modeling, verification, testing and optimization of hardware/software systems. He published more than 180 papers in the EDA field; two of them received the "best paper awards" respectively at IEEE EURODAC'96 and IEEE DATE'99.