Energy Management of a Set of Sensor Nodes at Application Level using the LINC Middleware

Olesia Mokrenko

Air Liquide Medical Systems
Parc de haute technologie,
6 rue Georges Besse,
F-92182 Antony, France
Email: olesia.mokrenko@airliquide.com

Suzanne Lesecq, Maria Isabel Vergara-Gallego

Univ. of Grenoble Alpes
CEA, LETI, MINATEC Campus
17 rue des Martyrs,
F-38054 Cedex 9, France
Email: FirstName.LastName@cea.fr

***Abstract*—Optimization of the energy consumption of sensor networks is traditionally performed either at the sensor node level or at the network level. However, more energy savings can be obtained if the application that makes use of the sensor nodes is considered. In order to achieve such extra energy gains, control theory can be applied. This paper summarizes control strategies implemented to minimize the energy consumption of a set of sensor nodes, while ensuring the application Quality of Service (QoS), this latter being mainly expressed with a minimum number of samples that must be available at the application level at each sampling time. With the Control strategies proposed, the sensor network lifetime is increased compared to the case without control strategy at application level. The control strategies have been implemented and evaluated on a real testbed composed of heterogeneous sensor nodes, and using the LINC middleware for node coordination.**

Keywords–Energy management; control strategy; middleware.

I. INTRODUCTION

Sensor Networks, and especially their wireless version (WSNs), have been a hot research topic for the last decades [1]. The optimization of energy consumption, in particular when the nodes are powered by batteries, is mandatory if the goal is to deploy sensor nodes that will run autonomously for years, without battery changes. This goal can be achieved if the power consumption is drastically decreased at all the "system" levels, from the sensor node itself to the communication protocols, but also at the application level.

At sensor node level, several improvements already exist, ranging from the design of novel radio technologies [2] to microcontroller architectures [3], and energy management methodologies [4]. Many approaches have been proposed at the different layers of the communication stack to increase the network lifetime [5]. They mainly minimize the node active period using duty cycle, optimize routing protocols, and reduce the quantity of transmitted data. However, the WSN lifetime can be extended if the energy consumption is reduced not only at the node level but also at a more global level, i.e. including the applications running on top of the WSN.

A trade-off between performance (or application QoS) and energy consumption must be found. A promising solution to reach this trade-off is to rely on Control Theory. From a model of the system (here a WSN system), possibly taking into account constrains, Control Theory can ensure the application QoS while minimizing the energy consumption of the whole WSN. This control objective can be naturally expressed as an optimization problem. The associated control law will then require a bit of information from the nodes and protocol layers (e.g., energy consumption of the nodes, remaining energy in the batteries of the nodes, the work each node has to perform) in order to manage the nodes and the whole system. To our knowledge, most of the work relying on Control Theory for the minimization of energy consumption in WSNs are related to managing the transmission power of the sensor nodes. These mechanisms are applied at the node level, where no knowledge regarding the global state of the network is required. Moreover, the application constraints are not taken into account. Note that [5] reviews various energy conservation schemes but none of them seems grounded in control theory. [6] proposes a classification of power control mechanisms in WSN, based on Control Theory and actually related to power control protocols only. The present paper details the implementation, on a real test-bed, of power control strategies at application level based on Model Predictive Control (MPC) and on a Hybrid Dynamical System (HDS) approach. The objective of both strategies is to minimize the energy consumption of the whole WSN while the application QoS is fulfilled. This problem is naturally expressed as an optimization one with constraints, which boils down to MPC. However, the system to be controlled (i.e., the WSN) is hybrid, with evolution of the battery charge driven by differential equations while the state of the sensor nodes is discrete (e.g., on, off, sleep modes). Therefore, a HDS approach is also a good candidate for the WSN power control. For the selected test-bed, the MPC strategy described in [7],[8], doubles the WSN lifetime when compared to the implementation without control at application level. Theoretical details regarding the HDS strategy can be found in [9]. Preliminary results show that, in simulation, the lifetime is extended by a factor slightly smaller than 2, which is logical due to the sub-optimal solution provided by the HDS approach.

The main goals here are to describe how the gap between theory and implementation has been closed, and demonstrate and validate the theoretical results. Thus, the control strategies will not be described in details, and the reader can refer

 ISBN: 978-1-61208-514-2

to [7],[8],[9] for details regarding the control approaches. Two communication technologies are deployed on the test-bed, leading to a heterogeneous network. This shows that the proposed strategies are independent from the communication technologies. The implementation is based the LINC [10] coordination middleware. Thanks to its resource-based approach and transactional guaranties, LINC makes it possible for the global controller to take decisions according to the current state of the sensor nodes, and to force the WSN to behave as expected at the application level.

The paper is organized as follows. Section II presents a description of the sensor network considered and the proposed application. Section III describes how the network has been implemented and integrated with the power controllers (MPC and HDS) at application level. Section IV summarizes the implementation results. Finally, Section V discuses the related works and Section VI summarizes the main findings and gives future work directions.

II. System description

Consider a WSN that has been deployed to monitor environmental parameters in a given zone in a building. The parameters typically monitored are the temperature, the CO_2 and humidity levels. The sensor readings are sent by the sensor nodes to a sink node that will basically interact with building appliances, e.g., the heating system or the air conditioning units. Actually, measurements are collected to feed the Building Automation Systems that will control the temperature or the ventilation in the given zone. These correspond indeed to the application level. The present work deals with the collection of enough measurements for the application to perform its task (e.g., temperature control, ventilation control) adequately. To reach a nominal behavior at the application level, a given application Quality of Service (QoS) must be reached. Here, the application QoS refers to the minimum amount of information required for the correct functionality of the application. It is defined with the sampling frequency of the nodes and the quantity and quality of sensor readings.

In the present work, the sensor nodes are supposed to be densely deployed in the zone, leading to information redundancy. Thus, the minimum number of nodes needed to ensure the application QoS is smaller than the total number of nodes deployed. It is therefore possible to switch off some nodes, leading to energy savings and network lifetime extension. Moreover, a node can be replaced by another one when, for instance, the first one runs out of energy.

The sensor nodes may differ from:

- *the available energy:* sensor nodes are powered by batteries with possibly harvesting systems to harvest energy. As a consequence, the nodes can have different available energy;
- *the energy consumption:* to read a given environmental parameter, the sensor nodes may require different energy budgets, depending on their hardware;
- *heterogeneity:* sensor nodes can be heterogeneous regarding communication protocol, data format and/or hardware characteristics.

As a consequence, the choice of the active nodes at a given instant of time is not straightforward. Moreover, nodes may become *Unreachable*) without notice. This disconnection may be caused by energy shortage, communication troubles, or hardware failures. Nodes may also "re-enter" the network when they leave the *Unreachable* state if the *Unreachable* condition is no more valid. As multiple hardware platforms and communication protocols can be deployed, it is necessary to rely on a middleware layer to support heterogeneity. Such a layer abstracts the controller from the communication protocols, data formats, or other low level matters.

Fig. 1 illustrates the data collection system, split in four levels: I) the *sensor nodes*, II) the *communication infrastructure*, III) the *data collection and synchronisation* level and IV) the *Controller*. These levels are now detailed.

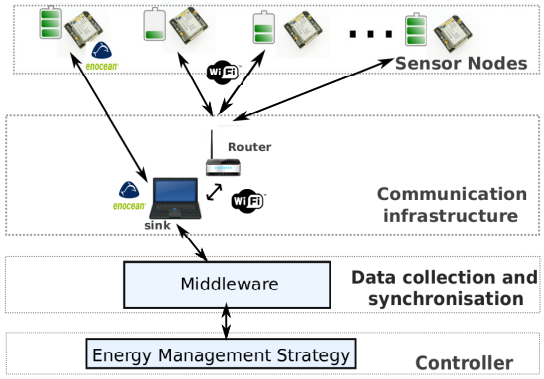

Figure 1. WSN System levels

A. Sensor nodes

Sensor nodes are usually split in four main parts, namely, computation, communication, sensing and power supply [11]. For each node, different power modes are defined, corresponding to a combination of the state for each unit in the node. The energy consumption of the node in a given power mode is given in the node datasheet provided by the manufacturer. The activity of the sensor nodes (i.e., sensing, computing, communication) is usually "duty cycled": the node periodically wakes-up, senses environmental parameters, processes and transmits these data, and finally enters the sleep mode (i.e., all units are disabled, waiting for a timer event). The node consumption is related to the duty cycle.

The energy management strategies implemented here assign a *functioning mode* to each *Reachable* node. Each functioning mode is defined by a given period for the sensing and communication tasks. Hence, all nodes in the network are duty cycled, and their energy consumption depends on their duty cycle for communication and sensing. Basically, the controller (that implement either the MPC of a HDS approach) combines information received from the nodes regarding their remaining energy and the application QoS requirements, to decide the most suitable functioning mode for each *Reachable* node.

Information regarding the remaining energy of the nodes is sent on a periodic basis to the controller. Thus, a minimum communication duty cycle is required for the proper functionality of the controller. Note that the remaining energy must be estimated because it cannot be measured directly.

 ISBN: 978-1-61208-514-2

B. Communication architecture

In order to exchange data between the nodes and the sink, a communication topology is chosen. Here, a star topology is considered and the sensor nodes communicate directly with the sink through a router. Hence, the controller can choose to increase the communication period of one node, without affecting the communication with the other ones.

Note that a cluster-tree topology can be easily implemented. For this topology, the cluster coordinators are responsible for forwarding information from nodes in the cluster. Thus, coordinators are duty cycled and synchronized with the associated nodes for data reception and forwarding. Hence, to communicate with all the associated nodes, the wake-up (or communication) period of the coordinator must be at most the shortest communication period of all the associated nodes. Therefore, the controller can change the functioning mode of all end-nodes in the network. The functioning mode of the coordinators are then set accordingly. Besides, further parameters have to be taken into account by the controller. For example, the number of hops required to send a packet to the sink, which is proportional to the communication energy cost, must be considered. All these factors can be easily integrated in the controller. Nodes must inform their role in the network (coordinator or end node) and the number of hops to the sink. The controller will take into account all these aspects when control decisions are provided, as in the star topology case.

More complex topologies, such as mesh networks, require the controller to know the current topology and routing information, which is highly dynamic. Indeed, the controller cannot arbitrarily change the functioning mode of the nodes because it can break the routing and topology maintenance mechanisms. Thus, the integration with a mesh network requires extra extension of our control strategies.

C. Data collection and synchronization

This third layer is in charge of data collection and synchronization of the sensor nodes. It calls the controller when its mandatory inputs are “ready”. These tasks are achieved through the LINC [10] coordination middleware.

LINC uses an associative memory [12] implemented as a distributed set of bags. This offers a decoupling in space and time between data producers (here, the sensor nodes) and data consumers (here, the controller). LINC has been successfully used in various application domains. It currently supports around 30 communication technologies. The associative memory also provides an abstraction layer to present all the measurements in the same format to the controller.

To ensure the expected application QoS, the sensor nodes communicate with the sink at a given time period (depending on their functioning mode) that depends on the application domain. Between two communication instants, the external timer of the sensor node is activated according to the desired period. Then, the nodes enter the sleep state, waiting for a timer interrupt. However, the node oscillators may experience drift over time. Therefore, if the clocks are not re-synchronized, some nodes may wake up outside of the time-window accepted by the sink. Moreover, if they wake up too early, their measurements will be outdated when used by the controller. If they wake up too late, the controller will not take their readings into account as the nodes will be considered *Unreachable*.

To avoid synchronization issues, LINC ensures that the clocks of the nodes are synchronized, with a synchronization frame periodically exchanged. The synchronization period depends on the type of oscillators being implemented in the nodes (the more precise the oscillator, the longer the synchronization period). This parameter can be determined empirically or according to the characteristics of the nodes. The synchronization procedure is completely independent from the controller. Thus, this latter does not need to take care of synchronization burden. It only processes the measurements currently received from the *Reachable* nodes.

The time/space decoupling and the abstraction offered by LINC have been particularly useful here. Indeed, it is straightforward to ensure that the latest measurements are available in the associative memory. The measurements are simply added to a dedicated bag whenever this is required (sampling/ synchronization periods). The controller is called when the current state of the nodes is known. The LINC application waits during a configurable time period to receive information from nodes. If no information is received after this time, the node is considered *Unreachable*.

D. Control design

The controller is based either on a Model Predictive Control (MPC) or on a HDS approach. Details on these strategies are provided in [7],[8],[9] and will not be reported here.

The energy consumption of each node during a sampling period depends on the functioning mode that is assigned by the controller. To increase the network lifetime (i.e., the time the network can ensure the application QoS), the controller must choose the functioning mode of each mode (which corresponds to the “control value”) in order minimize the energy consumption of the whole set of nodes while the application QoS is met. This objective is indeed an optimization problem that can be solved using Model Predictive Control. Actually, at each control sampling time, MPC minimizes a given cost function (possibly under constraints) over a control horizon in order to fix the control values. Due to the discrete nature of the functioning mode (on, of, sleep, unreachable), the optimization problem is a Mixed Integer Quadratic Programming (MIQP) one, the optimization variables being both boolean values (control value) and positive real values (available energy in the node battery) with bounds. Moreover, a set of constraints must be taken into account. The first subset is related to the remaining energy in the batteries that must be strictly positive to avoid battery damages (lower bound). Moreover, the remaining energy cannot be infinite and a maximum value cannot be exceeded. The second subset of constraints expresses that each node can be in a unique functioning mode. The last subset expresses the application QoS. The reader can refer to [7] where theoretical details are provided.

The control objectives can also be achieved using a Hybrid Dynamic System approach. Actually, the system under control is hybrid *per se*: the remaining energy in the node battery is conducted by differential equations while the functioning modes are discrete values. Moreover, a given node will experience “jumps” from one mode to another one, depending

 ISBN: 978-1-61208-514-2

TABLE I. POWER CONSUMPTION OF FLYPORT PLATFORM AND ENOCEAN TRANSCEIVER (Supply Voltage 3.3 V) [13], [14]

Mode	Power	Remarks
Wi-Fi connected	162, 70 mA	MCU *on* / Wi-Fi *on*, connected to access point
Wi-Fi not connected	39, 75mA	MCU *on* / Wi-Fi *on* but not connected
EnOcean *Rx*	61.21 mA	No Wi-Fi
EnOcean *Tx*	52.21 mA	No Wi-Fi
Hibernate	28,21 mA	MCU *on* and Wi-Fi *off*
Sleep	1,44 mA	MCU *off* and Wi-Fi *off*

on the controller decisions. This will modify the differential equation that models the node energy consumption. The reader can refer to [9] where theoretical details are provided.

Both controllers have been first designed in the Matlab environment, and evaluated in simulations. The numerical values used in simulation (e.g., maximum energy in the node batteries, energy consumption in each functioning mode) have been extracted from datasheets [13]. Then, the controllers have been written in python, and optimized to reach an efficient implementation in terms of computational time.

III. IMPLEMENTATION DESCRIPTION

The main objective of this work was to validate the theoretical and simulation results obtained with the energy management strategy at application level, on a real test-bed. This latter is now described, along with the data collection approach.

A. Test-bed description

The hardware test-bed consists of one sink, one router, and 6 sensor nodes spread over a given zone. The sensor nodes are the *Openpicus* Flyport Wi-Fi 802.11g [13] with either Wi-Fi or EnOCean TCM-310 [14] transceivers, leading to a heterogeneous network. Table I summarizes the power consumption of the Flyport platform. The *Openpicus* Flyport Wi-Fi 802.11g platforms embed a Microchip PIC 16-bits processor with a Wi-Fi radio module and ready-to-use protocol stacks. *Openpicus* provides a FreeRTOS-based framework implementing the CSMA-CA MAC protocol and the TCP transport protocol. Applications are written as FreeRTOS tasks.

When using Wi-Fi, *Openpicus* nodes form an infrastructure topology: they connect to one access point or router. This latter forwards all the received packets from/towards the sink. For the EnOcean protocol, an EnOcean transceiver is connected to the GPIOs of the Openpicus platform. This transceiver exhibits very low-power serial communication, that permits the exchange of very short frames. The EnOcean node communicates directly with the sink, this latter making use of an EnOcean USB dongle module.

B. Functioning mode definition

Two functioning modes and an *Unreachable* state are defined for each node, for both the Wi-Fi and the EnOcean transceiver:

- in *Mode 1* (*Active*), both the communication and sensing tasks are activated periodically. The sensing period is equal to the communication duty cycle $T_s = 1min$;
- in *Mode 2* (*Standby*), the node sensing unit is disabled. The node wakes-up periodically, as requested by the controller, to report to the sink its battery state-of-charge and to receive its functioning mode for the next period of time as computed by the controller. The communication period of a node in this mode is equal to the controller period $T_c = 1hour$;
- nodes enter the *Unreachable* state when they are unable to communicate with the sink due to network issues, hardware damage, or lack of energy.

When a node transmits a frame to the sink (e.g., sensed value, available energy) it waits for an answer. The answer can be an acknowledgement, a resynchronisation frame, or a command frame asking the node to change its functioning mode.

C. Application QoS

The application QoS is expressed as the minimum number of measurements that must be provided by the WSN to the application at each control sampling time. It is related to the *mission* that must be performed by the controller.

In the scenario implemented, d_1 nodes must be assigned to the *Active* mode in order to meet the mission. The mission can also change dynamically, i.e., depending on a time schedule or on external events. This dynamic feature permits following the needs of the application over time. In the present setup, we consider that during working hours, $d_1 = 3$ nodes must be in *Mode 1* while during the night periods d_1 is equal to 1.

The lifetime of the WSN is defined as the period of time during which the mission can be fulfilled. This corresponds to the time when the number of reachable nodes becomes smaller than the minimum number of required active nodes d_1.

D. Data collection

To collect data from the sensor nodes, the nodes must be encapsulated in LINC. This is made easy using the PUTUTU framework [15] that provides generic LINC objects to speed up integration of sensor and actuator technologies. Fig. 2 illustrates the LINC application put in place. The application is composed of three LINC objects all running in the sink:

- the *openpicus_wifi_object* acts as a TCP server that listens for connections from Wi-Fi nodes. It stores measurements and battery information sent by these nodes;
- the *openpicus_enocean_object* opens a serial connection with the EnOcean dongle to communicate with the EnOcean nodes. It stores the measurements and battery information sent by these nodes;
- the *controller_object* collects information about the battery status of all reachable nodes (from the two other objects). It periodically calls the controller to get the new functioning mode for each node. The updates are propagated to the `openpicus_wifi_object` and the `openpicus_enocean_object` that send the new functioning mode to each node if it has changed.

 ISBN: 978-1-61208-514-2

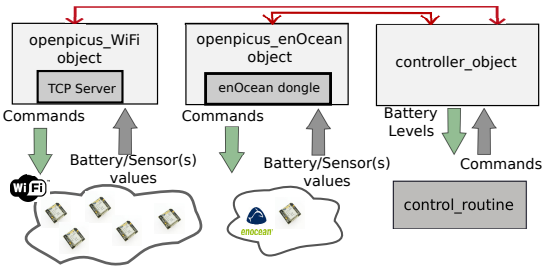

Figure 2. LINC Application for integration

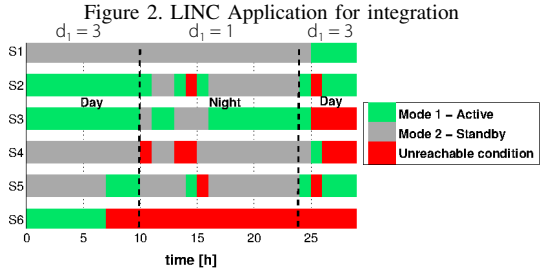

Figure 3. Functioning mode of each sensor node vs. time. Example with MPC strategy

The first time the nodes communicate with the sink, their clock is synchronized with the sink clock and keep synchronized according to the synchronization period. This latter can be adjusted dynamically. Empirical experiments have shown that a synchronization period of approx. $6hours$ is appropriate for this implementation. This synchronization period leads to a maximum clock drift of a couple of seconds, which is acceptable given the dynamics of the application.

IV. Experimental Results

An experiment of 30 $hours$ duration was run to evaluate both power control strategies at application level. The experiments started at $T_0 = 8\ a.m.$ S6 is the EnOCean node, the other ones being Wi-Fi. During working hours (resp. night period), 3 (resp. 1) sensors must be *Active*. Fig. 3 shows the functioning modes of the sensor nodes during the experiment when the MPC strategy is implemented. As can be seen, the mission is fulfilled during all the experiment duration. It can be observed that S6 became *Unreachable* after 6 hours, when its battery reached the minimum energy level. S4 was disconnected twice from the network at time10 hours during 1 hour, and at time 14 hours during 2 hours. The disconnection is due to radio channel perturbations. The same phenomenon happens with nodes 2 and 5. When a node falls in the *Unreachable* state, it is no longer taken into account by the controller. When the controller receives again information related to the remaining energy of this sensor node, it adds the node again in the set of nodes to be controlled. Note that the nodes do not possess "wake-up-on-event" capability. Therefore, if an active node becomes unreachable during a control period T_c, the controller will switches a node from mode 2 to mode 1 when T_c is elapsed. This means that we may experience periods of time no longer than T_c when the mission is not strictly speaking fulfilled. If the nodes could be woken up at any time by an external mechanism, before the end of the duty cycle, this issue would be solved.

In the present scenario, harvesting systems are not integrated in the sensor nodes. When available, they can help a node re-enter the "game" when it has regained enough energy to communicate and be placed by the controller in the *Active* or *Standby* mode.

TABLE II. SCALABILITY OF THE MPC AND HDS APPROACHES

Number of nodes	6	18	54
MPC Computational Time $[sec]$	0.051	0.052	0.053
HDS Computational Time $[sec]$	0.004	0.043	0.375

Using the MPC (resp. HDS) approach, the lifetime is increased up to $36,7\%$ (resp. 30.8%) when compared to the "basic" control. This latter corresponds to the situation where only the mission is considered, without any concern regarding energy consumption.

The scalability of the MPC (when a MILP solver is used) and HDS approaches has been analyzes. Table II shows the controller computational time obtained in the Matlab environment (on an Intel Xeon Processor E5620 with 12MB of cache, 2.50 GHz) for 6, 18 and 54 sensor nodes. Notice that the HDS computational time becomes higher that the MPC one due to state space explosion.

V. Related Work

To the authors' knowledge, most of research works related to power management in WSN focus on the protocol stack [5], [6] or on the node hardware optimization. The application needs are seldom taken into account in the energy/power management strategies [16]. This latter proposes a Dynamic Power Management (DPM) strategy at node level that takes into account application constraints to keep the sensor node, as much as possible, in sleep or idle mode without losing the real time responsiveness of the application. The DPM strategy is based on a hybrid automaton. It is implemented within the node. Its main advantage compared to our approach is that the control strategy is fully distributed. However, the DPM strategy does not have knowledge of the WSN state and of the changes in the application needs. Moreover, having the DPM strategy embedded within the node induces overconsumption. Note that there is no real implementation of the DPM strategy proposed in [16]. The present paper relies on a middleware layer to apply the control approach. The controller collects information from the nodes and it can fix some parameters (e.g., the functioning mode of each node). Middlewares and frameworks have been previously proposed with similar purposes. PyFUNS [17] is a framework that modifies network parameters according to the application. In PyFUNS, applications are written as python scripts. Then, the framework calibrates the network for energy efficiency. However, it is suitable only for nodes running the ContikiOS. The architecture described here may support any operating system and communication stack thanks to LINC. MILAN [18] is a middleware that continuously controls the network functionality according to the application demands. The implementation of such mechanism is very complex given the huge number of parameters that must be taken into account. Moreover, it seems difficult to integrate a control strategy with MILAN where several assumptions are made on the WSN behaviour. Other coordination middlewares, also using associative memory, have been used for WSN, such as Agimone [19], TeenyLIME [20] or Agilla [21]. However, they do not offer the same properties as LINC (e.g., distributed transactions, support of many

 ISBN: 978-1-61208-514-2

protocols and integration frameworks). Moreover, these works focus on developing applications.

VI. Conclusions

This paper presents the implementation of control strategies (MPC and HDS) for global energy management of a WSN. The implementation uses the LINC coordination middleware. The different aspects taken into account for the proper implementation, such as data collection and synchronisation, are described Experimental results on a star topology are reported. The functioning mode of each reacheable node is set according to decisions taken by the global controller whose objective is to minimize the energy consumption of the whole set of sensor nodes while ensuring the application Quality of Service. Simulation results show that the control strategies are scalable. The control approach is based on Model Predictive Control. This permits to add new constraints or control objectives if needed. LINC has been used in various application domains (e.g., smart buildings, smart cities or smart parking). Hence the proposed approach may be extended to other application domains.

Future work directions include the implementation of the control strategy on a WSN with a more complex topology, such as mesh networks. Such implementation will require a deeper knowledge about the network status, i.e., routing information and current topology.

Acknowledgment

This work has been partly funded by the Artemis ARROWHEAD nb332987 and the H2020 TOPAs nb676760 projects.

References

[1] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," Computer Networks, vol. 52, no. 12, 2008, pp. 2292 – 2330.

[2] F. D. Hutu, A. Khoumeri, G. Villemaud, and J.-M. Gorce, "A new wake-up radio architecture for wireless sensor networks," EURASIP Journal on Wireless Communications and Networking, Oct. 2014, p. 177.

[3] M. Hempstead, M. J. Lyons, D. M. Brooks, and G.-Y. Wei, "Survey of hardware systems for wireless sensor networks." J. Low Power Electronics, 2008, pp. 11–20.

[4] Y. Akgul et al., "Power management through dvfs and dynamic body biasing in fd-soi circuits," in Proceedings of the 51st Annual Design Automation Conference. ACM, 2014, pp. 1–6.

[5] G. Anastasi, M. Conti, M. D. Francesco, and A. Passarella, "Energy conservation in wireless sensor networks: A survey," Ad Hoc Networks, vol. 7, 2009, pp. 537 – 568.

[6] N. Pantazis and D. Vergados, "A survey on power control issues in wireless sensor networks," IEEE Communications Surveys, vol. 9, 2007, pp. 86 – 107.

[7] O. Mokrenko et al., "Dynamic power management in a wireless sensor network using predictive control," in 40th Annual Conference of the IEEE Industrial Electronics Society, 2014.

[8] Mokrenko et al., "Design and implementation of a predictive control strategy for power management of a wireless sensor network," in IEEE European Control Conference, 2015.

[9] O. Mokrenko, C. Albea, L. Zaccarian, and S. Lesecq, "Feedback scheduling of sensor network activity using hybrid dynamical system approach," in 54th IEEE Conference on Decision and Control, 2015.

[10] M. Louvel and F. Pacull, "Linc: A compact yet powerful coordination environment," in Coordination Models and Languages, ser. Lecture Notes in Computer Science. Springer, 2014, pp. 83–98.

[11] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," Computer networks, 2002, pp. 393–422.

[12] N. Carriero and D. Gelernter, "Linda in context," Commun. ACM, vol. 32, 1989, pp. 444–458.

[13] openpicus, "http://www.openpicus.com, 2015." [Online]. Available: http://www.openpicus.com

[14] enocean, "www.enocean.com." [Online]. Available: www.enocean.com/en/enocean_modules/tcm-310/

[15] F. Pacull et al., "Self-organisation for building automation systems: Middleware linc as an integration tool," in IECON 2013-39th Annual Conference on IEEE Industrial Electronics Society. Vienna, Austria: IEEE, 2013, pp. 7726–7732.

[16] R. Passos, C. Coelho, A. Loureiro, and R. Mini, "Dynamic power management in wireless sensor networks: An application-driven approach," in Second Annual Conference on Wireless On-demand Network Systems and Services (WONS'05), pp. 109–118.

[17] S. Bocchino, S. Fedor, and M. Petracca, "Pyfuns: A python framework for ubiquitous networked sensors," in Wireless Sensor Networks. Springer, 2015, pp. 1–18.

[18] W. B. Heinzelman, A. L. Murphy, H. S. Carvalho, M. Perillo et al., "Middleware to support sensor network applications," Network, IEEE, vol. 18, no. 1, 2004, pp. 6–14.

[19] G. Hackmann, C.-L. Fok, G.-C. Roman, and C. Lu, "Agimone: Middleware support for seamless integration of sensor and ip networks," in Distributed Computing in Sensor Systems. Springer, 2006, pp. 101–118.

[20] P. Costa, L. Mottola, A. L. Murphy, and G. P. Picco, "Teenylime: transiently shared tuple space middleware for wireless sensor networks," in Proceedings of the international workshop on Middleware for sensor networks. ACM, 2006, pp. 43–48.

[21] Fok, Chien-Liang and Roman, Gruia-Catalin and Lu, Chenyang, "Agilla: A mobile agent middleware for self-adaptive wireless sensor networks," ACM Transactions on Autonomous and Adaptive Systems (TAAS), vol. 4, no. 3, 2009, p. 16.

 ISBN: 978-1-61208-514-2