Implementation of an Energy Management Control Strategy for WSNs using the LINC Middleware

Maria Isabel Vergara-Gallego, Olesia Mokrenko, Maxime Louvel, Suzanne Lesecq, François Pacull
Univ. of Grenoble Alpes
CEA, LETI, MINATEC Campus
{FirstName.LastName}@cea.fr

Abstract

Energy optimisation in Wireless Sensor Networks (WSNs) is traditionally done either at the sensor node level or at the network level. To obtain even more energy savings, the final application must be considered. Control theory offers promising solutions to tackle this challenge. This paper details an implementation of a control strategy to minimise energy consumption of a WSN, while ensuring the application Quality of Service (QoS). The application QoS is expressed with the sampling period for sensor nodes and a minimum number of samples that must be available at the application level at each sampling time. With a Model Predictive Control strategy, the lifetime of the WSN is doubled compared to the basic case. The control strategy has been implemented and evaluated on a real test-bed, formed of heterogeneous sensor nodes, thanks to the LINC coordination middleware.

Keywords

Energy Management, Middleware, Implementation, Model Predictive Control

1 Introduction

Wireless Sensor Networks (WSNs) have been a hot field of research for several decades [20]. A great deal of effort has been put on optimising energy consumption of single sensor nodes and/or of the whole WSN. The grail is to implement distributed applications that will run autonomously for years, without battery changes.

To make this a reality, power consumption must be drastically reduced at the sensor node level. Several solutions, that tackle this objective already exist at the design time of the node hardware and/or software. Regarding hardware design, solutions can go from the design of novel radio technologies [12] to microcontroller architectures [11], and energy management methodologies [1]. At the software level, in the literature and industry, a large number of approaches and algorithms are proposed at different layers of the communication stack to increase the network lifetime [3]. These approaches try to minimise the nodes active period (using duty cycle), optimise routing protocols, reduce the quantity of transmitted data, etc. However, maximising the WSN lifetime boils down to reducing the energy consumption not only of nodes (node level) or of one part of the network (network level). Energy must be considered at the global level, i.e. including the applications running on top of the WSN.

A trade-off between performance (or application Quality of Service (QoS)) and energy consumption must be found. A promising solution to find this trade-off is to rely on Control Theory. From a model of the system (here a WSN system), possibly with constrains, Control Theory can ensure the application QoS while minimising the energy consumption of the whole WSN. This control objective can be expressed with a cost function to be minimised under constraints. The associated control law will then require a bit of information from the nodes and protocol layers (for instance, 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 minimisation 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 [3] reviews various energy conservation schemes but none of them seems grounded in control theory. [18] proposes a classification of power control mechanisms in WSN, based on control theory. These works are actually related to power control protocols only.

The present paper details the implementation, on a real test-bed, of a control strategy based on Model Predictive Control (MPC). The objective of this strategy is to minimise the energy consumption of the whole WSN and, at the same time, to ensure the application QoS. For the selected test-bed, the MPC strategy, which is described in [15], extends the WSN lifetime by a factor of 2, compared to the situation where no MPC control is implemented. Theoretical foundation and preliminary results on a real test-bed can be found in [14]. The main goal of the present paper is to detail how it was possible to bridge the gap between theory and imple-

EWSN'16, February 15–17, 2016, Graz, Austria.

mentation, demonstrate and validate the theoretical results. In addition, two communication technologies are deployed, leading to a heterogeneous network. The implementation is based on LINC [13], a 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 WSN, and to force the WSN to behave as expected.

The paper is organised as follows. Section 2 presents a description of the WSN considered here and the application. Then, Section 3 describes how the WSN has been implemented and integrated with the global controller. Section 4 presents the implementation results. Finally, Section 5 discuses the related works and Section 6 gives some global conclusions and future work directions.

2 System description

The system considered in this paper is a WSN deployed to monitor an environmental parameter in a particular area, using several sensor nodes. For monitoring applications in buildings, typical monitored parameters are temperature, CO_2 level (indoor or outdoor), and humidity. The values acquired by the sensor nodes are sent to a sink node that will, for instance, interact with the air conditioning unit. Though the system may contain a control part (i.e., commands can be sent to actuators), this work focuses only on the monitoring part. To reach a nominal behaviour at the application level, a certain application Quality of Service (QoS) must be provided. In this context, the application QoS refers to the minimum information required for the correct functionality of the application, and it can be defined from the sampling frequency of the nodes, and the quantity and quality of sensed values. Note that the sensor nodes are supposed to be densely deployed in the sensor field, leading to a certain degree of redundancy. As a consequence, when the minimum number of nodes needed to ensure the application QoS is lower than the total number of nodes in the network, it is possible to switch off some nodes. In this way, the network lifetime can be extended. In addition, a node can be replaced by another when, for instance, the first one runs out of energy. Nodes sensing the same environmental variables may differ according to:

- *available energy:* sensor nodes are powered by a battery and/or harvesting system(s), hence at every instant they have different available energy (i.e., due to different battery sizes, harvesting periods, battery ageing);
- *energy consumption:* to provide the same type of measurements, different sensor nodes may require different energy budgets;
- *heterogeneity:* sensor nodes may be heterogeneous in terms of communication protocol, data format and/or hardware characteristics.

Hence, deciding which node must be active at a given instant of time is not straightforward. In addition, nodes can disconnect from the network (become *Unreachable*) without notice. Disconnection may be caused by energy starvation, communication problems, or hardware failures. Nodes may also join back the network when they leave the *Unreachable* state if they have regained enough energy thanks to harvesting systems, or if the communication is back.

Regarding heterogeneity, as multiple hardware platforms and communication protocols must be supported, it is necessary to rely on a middleware layer. Such a layer abstracts the controller from the communication protocol, data format, or any low level matter.

Figure 1 illustrates the monitoring system, which can be divided 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 detailed in the following subsections.

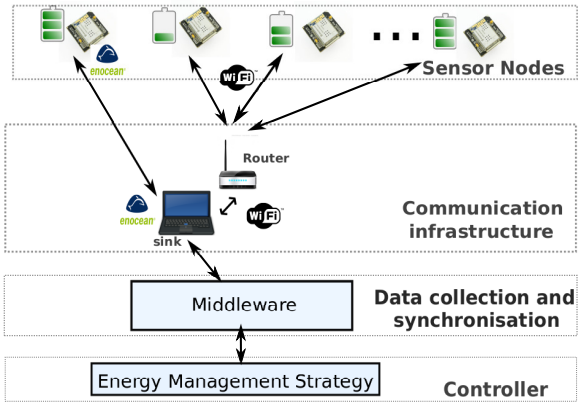

Figure 1. WSN System levels

2.1 Sensor nodes

Typical sensor nodes can be split in four main units, namely, computation, communication, sensing and power supply [2]. For each node, different power modes are defined. These power modes correspond 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 by the node manufacturer.

In a WSN, the activity of nodes (i.e. sensing, computing, communication) is usually "duty cycled": the node periodically wakes-up (i.e. the CPU is activated), senses an environmental value (i.e. the sensing unit is turned on), processes and transmits the information (i.e. the radio is activated), and finally enters the sleep mode (i.e. all units are disabled waiting for a timer event). The wake-up period (or duty cycle) determines the energy consumed by the node.

The energy management strategy implemented in this paper assigns different *functioning modes* to *Reachable* nodes. Each functioning mode is defined by a given period for 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 (i.e. each functioning mode is associated to a known energy consumption that depends on the duty cycle). Basically, the controller combines information received from the nodes regarding their remaining energy and the application QoS requirements, to decide the most suitable functioning mode for each node. For instance, if the controller objective is to minimise the energy consumption of a node, it can increase the communication period and deactivate the sensing activity (i.e. the sensing period is equal to infinity).

The controller requires information regarding the remaining energy of the nodes. This information must be sent on a periodic basis to the controller. Thus, a minimum communication duty cycle is required for the proper functionality of the controller. Notice that the nodes must embed an estimator of their remaining energy, since a sensor that measures the remaining energy in a battery does not exist.

2.2 Communication architecture

In order to exchange data with the sink, a communication architecture is defined. It determines, through a given topology, how nodes communicate with the sink. In this paper, a star topology is considered. 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 other nodes.

Note that a cluster-tree topology can be easily implemented and evaluated. For this topology, the cluster coordinators are responsible for forwarding information from nodes in the cluster. Thus, coordinators are duty cycled and synchronised with 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 energy cost of communication, 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 integrates this information to the constraints and takes decisions as for 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 as it can break the routing and topology maintenance mechanisms. Thus, the integration with a mesh network requires further work.

2.3 Data collection and synchronisation

This layer is responsible for collecting the sensed data, synchronising the sensor nodes, and calling the controller when the information is ready. These three tasks are achieved through the coordination middleware LINC [13].

LINC uses an associative memory [5] implemented as a distributed set of bags. This offers a decoupling in space and time between the data producer (the sensor nodes) and the data consumer (the controller). LINC has been successfully used in several application domains and currently supports around 30 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, sensor nodes need to communicate with the sink at a given period (according to their functioning mode). It can be of the order of seconds, minutes, hours, or days, depending on the application domain. Between two communication instants, the external timer of the sensor node is programmed according to the desired period. Then, the node enters the sleep state, waiting for a timer interrupt. Oscillators in sensor nodes are not perfect and they will drift over time. Therefore, if the clocks are not resynchronised, sensor nodes may wake up outside of the timing-window accepted by the sink. On the one hand, if the node wakes up too early, the measurement may be outdated when it is used by the controller. On the other hand, if the node wakes up too late, the controller will not take the values in consideration and may think the node is in the *Unreachable* state.

To avoid synchronisation problems, LINC ensures that the clocks of the nodes are resynchronised when needed. To do so, a resynchronisation frame is periodically exchanged with the nodes. The resynchronisation period depends on the type of oscillators being used by the sensor nodes (i.e., the more precise the oscillator, the longer the resynchronisation period). This parameter can be determined empirically or according to nodes' characteristics.

The resynchronisation procedure is completely independent from the controller. Thus, the latter does not need to take care of synchronisation. 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/ resynchronisation periods). The controller is called when the current state of the nodes is known. The LINC application waits during a configurable time to receive information from nodes. If no information is received after this time, the node is considered as *Unreachable*.

2.4 Control design

The controller is based on a Model Predictive Control (MPC) approach. In order to control energy saving in the WSN, the remaining energy in each sensor node i, $i = 1,...,n$, is modelled with a discrete-time linear model:

$$x_i(k+1) = x_i(k) + B_i u_i(k), \tag{1}$$

where $x_i(k)$ is the remaining energy in the battery of sensor node i at sampling time k. $B_i u_i(k)$ represents the energy that will be consumed during the time interval $[kT_c, (k+1)T_c]$. T_c is the control period.

$u_i = [u_{i1} \quad ... \quad u_{ih} \quad ... \quad u_{im}]^T$ is the control input. It is related to the functioning mode h, $h = 1,...,m$, of node i, where $u_{ih} \in \{0,1\}$. The control matrix is $B_i = [b_{i1} \quad ... \quad b_{ih} \quad ... \quad b_{im}]$. b_{ih} represents the amount of energy consumed by node i working in mode h.

The energy consumption of each node during this next period of time depends on the functioning mode assigned by the controller. The controller decisions tend to minimize the energy consumption of the whole set of nodes while the application QoS is met[1]. This decision comes from the minimi-

[1]Note that different control objectives can be added thanks to the flexibility of the MPC approach. For instance, the controller could balance the

Table 1. Power consumption of Flyport platform and EnOCean transceiver (Supply Voltage 3.3 V) [7,16]

Mode	Power	Remarks
Wi-Fi connected	162,70 mA	MCU *on* / Wi-Fi *on*, connected to access point
Wi-Fi not connected	39,75 mA	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*

sation of a cost function under constraints. Note that the optimisation problem is a Mixed Integer Quadratic Programming (MIQP) one, the optimisation variables being both boolean values (u_i) and positive real values (x_i) 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 interested reader can refer to [15] where theoretical details are provided.

The controller has been first designed in the Matlab environment, and evaluated in simulations. The numerical values used in the simulation (e.g. maximum energy in the node batteries, energy consumption in each functioning mode) have been extracted from datasheets [16]. Then, the controller has been implemented and optimised in python, leading to an efficient implementation in terms of computational time.

3 Implementation Description

The main objective of this work was to validate theoretical and simulation results. Thus, the proposed energy management approach is implemented on a real test-bed composed of heterogeneous wireless sensor nodes. The test-bed is now described, along with the data collection approach.

3.1 Test-bed description

The hardware test-bed considered here consists of one sink, one router, and 6 sensor nodes spread over a given area. The sensor nodes chosen for experimentation are the *Openpicus* Flyport Wi-Fi 802.11g [16] with Wi-Fi transceiver and with EnOCean transceiver TCM-310 [7], leading to a heterogeneous deployment. Table 1 gives 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. The 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.

3.2 Definition of the functioning modes

For the energy management strategy, two functioning modes and an *Unreachable* state are defined for each node, for both the Wi-Fi and the EnOcean transceiver:

- in *Mode 1* (or *Active* mode): both the communication and sensing tasks are activated periodically. The sensing period is equal to the communication duty cycle $T_s = 1$ *min*;
- in *Mode 2* (or *Standby* mode): the node sensing unit is disabled. The node wakes-up periodically, as requested by the controller, to report to the sink its battery state and to receive commands (i.e. its functioning mode for the next period of time) from the controller. The communication period of a node in this mode is equal to the controller period $T_c = 1$ *hour*;
- 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 (to send a sensed value or 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.

3.3 Definition of the application QoS

The application QoS is expressed as the minimum number of measurements that must be provided by the WSN to the application over a defined time period. Hereafter, this application QoS is called *mission*, which must be guaranteed by the controller. This latter assigns a functioning mode to the nodes, taking into account the nodes that are unreachable.

In the scenario, d_1 nodes must be assigned to the *Active* mode, in order to meet the mission, while minimising the overall energy consumption. Moreover, the mission can change dynamically, i.e. depending on a time schedule or on external events. This dynamic mission permits following the needs of the application during the system evolution. Here, we consider that during the working hours, when people are present in the office, the mission is defined with $d_1 = 3$ nodes in *Mode 1*, while during the night period, when there is nobody in the office, 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 time should be the time before the number of reachable nodes is smaller than the minimum number of required active nodes (d_1).

3.4 Data Collection

To collect data from the sensor nodes, the first step is to encapsulate the nodes in LINC. This is made easy by the PUTUTU framework [17] that provides generic LINC objects to speed up integration of sensor and actuator technologies. For instance, the EnOcean protocol was already integrated in the framework. Figure 2 illustrates the LINC application put in place. The application is composed of three

energy between nodes for fault tolerance

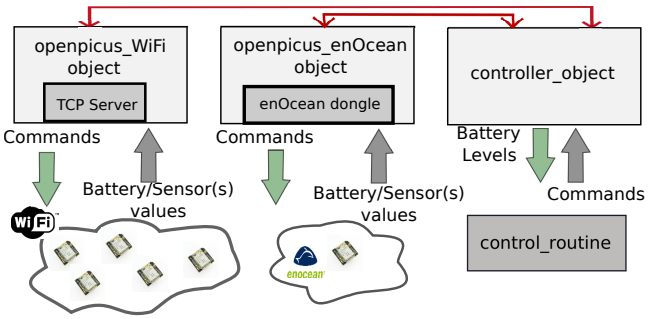

Figure 2. LINC Application for integration

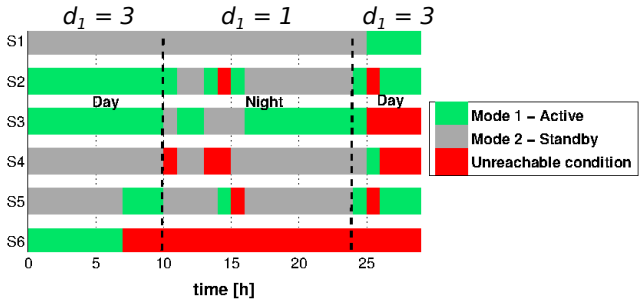

Figure 3. Functioning mode of each sensor node vs. time

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 sensed values 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 in the network. The updates are propagated to the `openpicus_wifi_object` and the `openpicus_enocean_object`. These latter send the new functioning mode to each node if it has changed.

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

4 Experimental Results

To evaluate the energy management strategy, an experiment of a duration of 30 hours was run. The experiment started at $T_0 = 8$ *a.m*. Figure 3 shows the functioning modes of the sensor nodes during the experiment. The EnOCean node is S6. We can see that the mission defined above is fulfilled during all the experiment: during the working hours, 3 sensors are *Active*, and during the night period, 1 sensor is *Active*.

In Figure 3 it can be observed that the node number 6 became *Unreachable* after 6 hours, when its battery reached the minimum energy level. Node 4 was disconnected twice from the network after 10 hours, during 1 hour, and after 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 solving the control problem. 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 (mode 1) becomes unreachable, due to the duty cycle $T_c = T_w = 1$ *hour* for the nodes in mode 2 (stand by), the controller will assign a node in mode 2 to mode 1 when the control period T_c is elapsed. This means that we can observe periods of time no longer than T_c when the mission is not fulfilled. If nodes could be woken up at any time by an external mechanism, before the end of the duty cycle, this issue would be solved.

Table 2. Scalability of the MPC approach

Number of nodes	6	18	54
Time [*sec*]	0.36	5.13	10.81

Note that, in the present scenario, nodes do not embed harvesting systems. In this latter case, when the battery of a node is drained, it can re-enter the "game" when it has regained enough energy to communicate and be placed by the controller in the *Active* or *Standby* mode.

Using the MPC approach, the fulfilling of the application QoS is increased up to $36,7\%$ compared to the "basic" control. The basic control presents the management of the node activity when only mission is ensured without any other parameter to control.

This scenario is also evaluated to demonstrate the scalability of the MPC approach. Table 2 shows the simulation results of one step computation time in Matlab environment for 6, 18 and 54 sensor nodes. We estimated the one step computation time with 54 nodes to 10.81 *sec*, this step needs to be computed every hour ($T_c = 1$ *hour*) in our example.

5 Related Work

To the authors' knowledge, most of the research works regarding power management in WSN focus on the protocol stack [3, 18], or on the optimisation of the node units. The application needs are seldom taken into account in the energy/power management strategies [19]. 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 a sleep or idle state without losing the real time responsiveness of the network application. The DPM strategy is based on a hybrid automaton. It is implemented within the node. The main advantage, compared to our approach, is that the control strategy is fully distributed. However, the DPM strategy does not have a global knowledge of the WSN state and the changes

in the application needs. Moreover, having the DPM strategy embedded within the node implies an overhead in the energy consumption of the node. Note that there is no real implementation of the proposed DPM strategy.

This paper relies on a middleware layer to apply the Control approach on a real WSN. The controller can then collects information from the nodes and sets some parameters of the network (i.e. the functioning mode of each node). Middlewares and frameworks have been proposed with similar purposes. For instance, PyFUNS [4] 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. Whereas the architecture described here, may support, thanks to LINC, any operating system and communication stack. MILAN [10] is a middleware that continuously controls the network's 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. It seems difficult to integrate Control Theory 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 [9], TeenyLIME [6] or Agilla [8]. However they do not offer the same properties as LINC (e.g. distributed transactions, support of many protocols and integration frameworks). Moreover, these works focus on developing applications for WSN where this paper integrates Control Theory. Hence, the guarantees offered by LINC are vital.

6 Conclusions

This paper has presented an implementation of a control strategy for global energy management of a WSN. The implementation is based on the LINC coordination middleware. This papers detailed the different aspects taken into account for the proper implementation, such as data collection and synchronisation.

Experimental results, on a star topology, show that the WSN is correctly integrated with the controller. Functioning modes of nodes are set according to decisions taken by the global controller, which minimises the energy consumption of the whole WSN while ensuring the application Quality of Service. The simulation results show that the control strategy is scalable. The control approach is based on Model Predictive Control. This permits to add new constraints or control objectives if needed. For instance one may way to spread the measurements for redundancy purpose. LINC has been used in several domains such as smart buildings, smart cities or smart parking. Hence the proposed approach may be extended to other applications or 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 such as routing information and current topology. Another interesting direction is to have a distributed controller and study the impact of distribution on the existing software architecture.

Acknowledge

This work has funded by the Artemis ARROWHEAD (grant 332987) and the H2020 TOPas project (grant 676760).

7 References

[1] Y. Akgul, D. Puschini, S. Lesecq, E. Beigné, I. Miro-Panades, P. Benoit, and L. Torres. Power management through dvfs and dynamic body biasing in fd-soi circuits. In *Proceedings of the 51st Annual Design Automation Conference*, pages 1–6. ACM, 2014.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer networks*, 2002.

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

[4] S. Bocchino, S. Fedor, and M. Petracca. Pyfuns: A python framework for ubiquitous networked sensors. In *Wireless Sensor Networks*, pages 1–18. Springer, 2015.

[5] N. Carriero and D. Gelernter. Linda in context. *Commun. ACM*, 32:444–458, 1989.

[6] 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*, pages 43–48. ACM, 2006.

[7] enocean. www.enocean.com.

[8] 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)*, 4(3):16, 2009.

[9] 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*, pages 101–118. Springer, 2006.

[10] W. B. Heinzelman, A. L. Murphy, H. S. Carvalho, M. Perillo, et al. Middleware to support sensor network applications. *Network, IEEE*, 18(1):6–14, 2004.

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

[12] 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.

[13] M. Louvel and F. Pacull. Linc: A compact yet powerful coordination environment. In *Coordination Models and Languages*, Lecture Notes in Computer Science, pages 83–98. Springer, 2014.

[14] O. Mokrenko et al. Design and Implementation of a Predictive Control Strategy for Power Management of a Wireless Sensor Network. In *IEEE European Control Conference*, July 2015.

[15] O. Mokrenko, S. Lesecq, W. Lombardi, D. Puschini, C. Albea, and O. Debicki. Dynamic power management in a wireless sensor network using predictive control. In *40th Annual Conference of the IEEE Industrial Electronics Society*, 2014.

[16] openpicus. http://www.openpicus.com, 2015.

[17] 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*, pages 7726–7732, Vienna, Austria, 2013. IEEE.

[18] N. Pantazis and D. Vergados. A survey on power control issues in wireless sensor networks. *IEEE Communications Surveys*, 9:86 – 107, 2007.

[19] 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)*.

[20] J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Computer Networks*, 52(12):2292 – 2330, 2008.