Mobile Netw Appl (2006) 11:469–485
DOI 10.1007/s11036-006-7194-1

Platform based design for wireless sensor networks

Alvise Bonivento · Luca P. Carloni · Alberto Sangiovanni-Vincentelli

Published online: 4 May 2006

Abstract We present a novel methodology for the design of interoperable Wireless Sensor Networks (WSN). The methodology is based on the principles of Platform Based Design (PBD). PBD is a meet-in-the-middle approach where the top-down refinement of a design specification meets with bottom-up characterizations of possible alternative implementations. The design space exploration is performed based on estimates of the performance of the candidate solutions so that the overall design process is considerably sped up as expensive re-designs are avoided and design re-use is favored. PBD is based on the rigorous definition of appropriate abstraction layers that are effective in shielding the drudgery of implementation details while allowing the important information to be taken into account. If each layer is formally specified, formal verification, refinement and synthesis are all possible. Yet while the overarching approach is general, the layers of abstraction and the accompanying tools can be (and in general, are) application dependent.

In this paper, we present three abstraction layers for WSNs and the tools that "bridge" these layers. We present a case study that show how the methodology covers all the aspects of the design process, from conceptual description to implementation.

A. Bonivento (✉) · A. Sangiovanni-Vincentelli
University of California at Berkeley
e-mail: alvise@eecs.berkeley.edu

A. Sangiovanni-Vincentelli
e-mail: alberto@eecs.berkeley.edu

L. P. Carloni
Columbia University,
Columbia
e-mail: luca@cs.columbia.edu

Keywords Platform based design · Sensor networks · Design automation

1. Introduction

Ad-hoc wireless sensor networks have the definite potential to change the operational models of traditional businesses in several application domains, such as the building industry [1], power delivery [2], and environmental control [3]. Sensor networks are already the essential backbone of the "ambient intelligence" paradigm, which envisions smart environments aiding humans to perform their daily tasks in a non-intrusive way [4].

This revolution has not escaped the attention of both academia and industry and has led to a flurry of activities such as the exploration of new applications and the development of new radio architectures, low-power wireless sensor nodes, low-date rate wireless protocols, and ad-hoc multi-hop routing algorithms. The creation of forms of interoperability between the myriad of hardware components and software protocols is essential for the full potential of these technologies to be achieved. In this context, a number of new wireless standards such as 802.15.4 [5] and Zigbee [6] are under development. Yet, it is our belief that these efforts created in a bottom-up fashion do not fully address the essential question of how to allow interoperability across the many sensor network operational models that are bound to emerge. In fact, different operational scenarios lead to different requirements, and hence different implementations, in terms of data throughput and latency, quality-of-service, use of computation and communication resources, and network heterogeneity. These requirements ultimately result in different solutions in terms of network topology, protocols, computational platforms, and air interfaces.

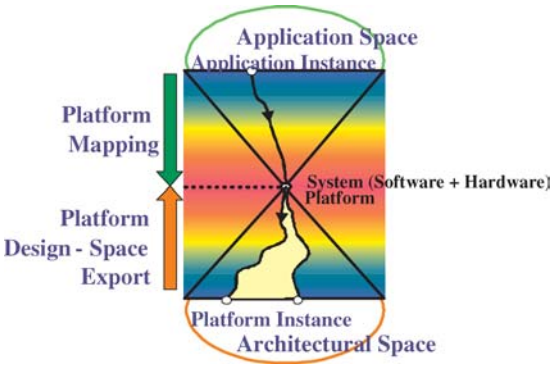

Fig. 1 The platform based design approach

To support true interoperability between different applications as well as between different implementation platforms and ensure scalability of sensor network technology, we advocate the use of a rigorous design methodology based on a set of appropriate abstraction layers. The proposed approach is an application of Platform Based Design (PBD) [8, 9]. PBD is a "meet-in-the-middle" design methodology, where system constraints are refined top-down, while implementation characteristics including performances such as delay and power consumption are abstracted bottom-up (see Fig. 1). The two parts are essential for selecting a good implementation via a design exploration phase that meets the constraints while estimating the performance of the candidate implementations. PBD relies on a clear identification of layers of abstraction, on a modeling strategy that captures uniformly functionality and architecture of the design, and on tools that map two contiguous layers, verify that the selected architectures satisfy constraints, and identify drawbacks and strengths of the design.

In this paper, we specialize the general PBD methodology to the WSN case. This methodology is based on three abstraction layers and the relative supporting tools. The top and the bottom layers have been introduced before; the intermediate layer is novel. The first layer is an application interface called Sensor Network Service Platform (SNSP) [7]. The SNSP defines a set of services available to the end user to specify the target application formally without dealing with the details of a particular network implementation. While the SNSP description suffices to capture the interaction between controllers, sensors and actuators, it is a purely functional description, which does not prescribe how and where each of these functions will be implemented. Hence, information such as communication protocols, energy, delay, cost, and memory size, are not available.

The second abstraction layer called Sensor Network Adhoc Protocol Platform (SNAPP) is presented here for the first time. The SNAPP defines a library of communication protocols and the interfaces that these protocols offer to the SNSP.

Once the communication protocol is selected, it must be implemented on a set of physical nodes. A description of the actual hardware platform is given by the Sensor Network Implementation Platform (SNIP).

Following the PBD paradigm, the process of mapping the SNSP description into a SNAPP instance and eventually into a SNIP instance goes through a set of steps. First, we need to ensure that the selected topology and communication protocol is capable of supporting the sensing, actuation and communication requirements implied by the application.

Once we derived the constraints on sensing, actuation and communication, we derive an abstraction of the physical layer properties of the proposed hardware platform (candidate SNIP instance), the bottom-up part of PBD, and select an adequate topology and communication protocol among the ones available in the SNAPP. Finally, we synthesize the parameters of the protocol so that that the given constraints are satisfied and energy consumption optimized.

Tools that help bridging between two different layers of abstraction can be built for particular domain of applications. In this paper, we focus on monitoring functions in an industrial plant. We first present Rialto,[1] a framework that takes the application described using the SNSP and derives a set of constraints for a SNAPP instance and then, a parametrized protocol SERAN [35], proposed by the research group of the authors for industrial applications, that provides a semi-automatic approach to protocol generation for this class of applications.

2. The abstraction layers

The definition of sockets in the Internet has made the use of communication services independent from the underlying protocol stack, the communication media and the various possible operating systems. In order to introduce a similar abstraction layer, we follow the approach first proposed in [7]. A properly defined application interface captures all the possible services that can be used by any sensor network application and supported by any sensor network platform.

2.1. The sensor network service platform

To perform its functionality, a controller (algorithm) has to be able to read and modify the state of the environment. In a WSN, controllers do so by relying on communication and coordination among a set of distinct elements that are distributed in the environment in order to complete three different types of functions: sensing, control and actuation. The

[1] Rialto is a famous Venetian bridge that links the two sides of the Canal Grande, hence the name of the tool that bridges two sides of an intellectual channel.

role of the Sensor Network Services Platform (SNSP) is to provide a logical abstraction for these communication and coordination functions. The SNSP decomposes and refines the interaction among controllers and between controllers and the environment into a set of interactions between controlling, sensing, and actuating functions. Hence, the services that the SNSP offers to the application are used directly by the controllers whenever they interact among each other or with the environment. This approach abstracts away the specific details of the communication mechanisms (routing strategies, MAC protocols, physical channel characteristics) thereby making possible for the application designer to focus on the task of developing the control algorithms for the WSN application.

In particular, as illustrated in 2 the SNSP is a collection of data processing functions (e.g. aggregation) and *I*/*O* functions (sensing, actuation) that cooperate in order provide the following services:

- *query service (QS)* used by controllers to get information from other components;
- *command service (CS)* used by controllers to set the state of other components;
- *timing*/*synchronization service (TSS)* used by components to agree on a common time;
- *location service (LS)* used by components to learn their location;
- *concept repository service (CRS)* which maintains a map of the capabilities of the deployed system and it is used by all the components to maintain a common consistent definition of the concepts that they agreed upon during the network operation.

The CSR is quite novel in the WSN community, but is deemed essential if a true ad-hoc realization of the network is to be obtained. The repository includes definitions of relevant global concepts such as the attributes that can be queried (e.g. temperature, pressure), or the regions that define the scope of the names used for addressing. It further allows collecting information about the capabilities of the system (i.e. which services it provides and at which quality and cost) and provides the application with a sufficiently accurate description. The repository is dynamically updated during the network operations. Access to the SNSP services is provided to the application through a set of primitives, combined in the *application interface (AI)*. The AI primitives can also make available to the application the relevant parameters that define the quality and the cost of the services.

2.2. The sensor network implementation platform

The Sensor Network Implementation Platform (SNIP) is a network of interconnected physical nodes that implement the logical functions of the application and the SNSP. A physical node is a collection of physical resources such as:

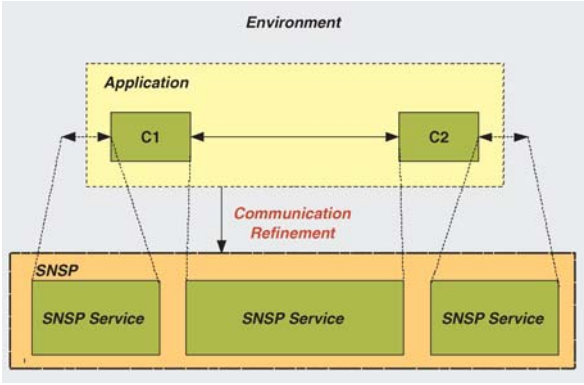

Fig. 2 The sensor network service platform

- clocks and energy sources;
- processing units, memory, communication, and *I*/*O* devices;
- sensor and actuator devices.

In particular, the main physical parameters of a node are:

- list of sensors and actuators attached to node;
- memory available for the application;
- clock frequency range;
- clock accuracy and stability;
- level of available energy;
- cost of computation (energy);
- cost of communication (energy);
- transmission rate (range).

2.3. The sensor network ad-hoc protocol platform

To choose the architecture of the SNIP and to map the functional specification of the system onto it are critical steps in sensor network design. To facilitate the process we created an intermediate level of abstraction called Sensor Network Ad-hoc Protocol Platform (SNAPP). The SNAPP is composed by a library of MAC and routing protocols that offer to the SNSP guarantees on latency, error rate, sensing requirements. These protocols are “parametrized protocols”, meaning that their structure is specified, but their working point is determined by a set of parameters. The value of these parameters is obtained as the solution of a constrained optimization problem, where the constraints are derived from the latency, error rate, sensing requirements of the application while the cost function is the energy consumption. The energy consumption is estimated based on an abstraction of the physical properties of the candidate hardware platform. The synthesis of these parameters represents the meet-in-the-middle phase of the PBD methodology when applied to the WSN domain.

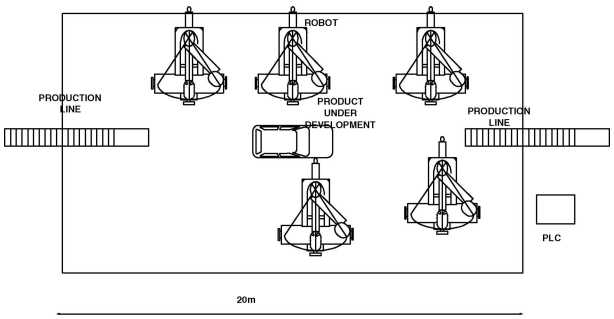

Fig. 3 Manufacturing cell

3. Industrial monitoring

To illustrate the proposed methodology, we study the application of WSN to the monitoring an automotive manufacturing plant.

In industrial automation control, sensors are deployed on the robots of a production line to monitor their state and prevent costly damage. Since these sensors are typically wired, the plant is characterized by a large amount of cables. Besides high maintenance and deployment costs, this solution also has a reverse impact on flexibility (the addition of new sensors and reconfiguration implies stopping the production line) and safety (human operators may trip on cables).

As illustrated in Fig.3, a typical manufacturing cell is an area within the manufacturing plant where several robots cooperatively work on the same production piece. Different types of sensors and actuators are deployed all over the cell and the behavior of these components is decided by a control routine running in the Process Loop Controller (PLC), a computer usually placed in the proximity of the cell. In particular, dedicated sensors report to the PLC the vibration and temperature patterns for each of the robots. The control routine evaluates the sensed data and takes decisions on the next action to preserve the proper working conditions. The control routine typically presents a high level of computational complexity, but a limited number of possible outcome decisions (i.e. move the robot up, or down, or switch it off). If the PLC notices that a particular robot shows high values of either the vibrations or the temperature parameter, it determines that the robot needs maintenance. Consequently, the PLC sends a command message to all the actuators to switch the robots off so that a human operator can perform the required maintenance before the robot creates expensive damage to the production line.

4. Rialto

In this section, we provide a summary of the basic steps of Rialto [34] a tool that targets WSN industrial control applications.

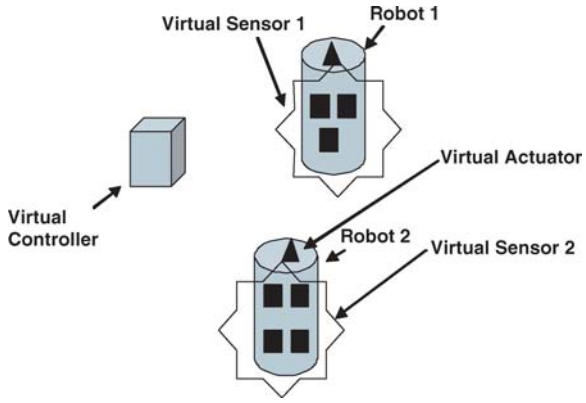

Fig. 4 Example of rialto model

To facilitate the presentation of Rialto, we first focus only on a subsystem of the industrial application presented in Section 3. This subsystem contains only two robots (Fig. 4).

Rialto supports only the subset of the services of the SNSP that is relevant for the chosen industrial domain, specifically: the query service, the command service and the concept repository service. Rialto allows the end user to specify a loose notion of the system topology in the concept repository service and to describe the control algorithm in terms of logical components, queries and commands. This description is independent from the particular communication infrastructure or hardware platform. The tool captures these specifications in a formal way and performs a state space exploration to analyze all the possible scenarios that the application may lead to. As a result of this exploration, it produces a set of design constraints that the communication links and hardware infrastructure must satisfy to ensure correct functionality of the network.

Once we derived the constraints on sensing, actuation and communication, and we have an abstraction of the candidate SNIP instance, we are ready to start the selection of an adequate SNAPP instance and synthesize the relative parameters to support the application.

4.1. Application description: Rialto model

In the Rialto model, applications are described in terms of logical components that communicate via queries and commands. Queries are requests for data and, as a consequence, each query is followed by a corresponding response. Commands are used to set some parameters or trigger some actions and do not necessarily need a response. The set of logical components, communication links, and queries and commands is called the Rialto Model (see Fig 4). The basic building blocks of the Rialto Model are actors and communication media. Actors use communication media to exchange tokens.

Tokens are the abstraction of queries and commands. A *query* is the formalization of the intuitive notion of data request such as: "sense vibrations from time T_i to time T_f with a sampling rate of X [samples/sec] and return to me the average within L seconds with a message error rate of 10^{-3}". Consequently, a query is composed of various fields whose parameter values define its content. One of these fields is the "attribute" that defines the quantity to be sensed (i.e. temperature, humidity, vibration, etc.), another one defines the required sampling rate, another one the time scope of the query (in the above example T_i and T_f), and so on. A *command* is the formalization of the intuitive notion of triggering an actuation such as: "switch the robot off from time T_i to time T_f; the command has to reach destination within L seconds and with a message error rate of X [samples/sec]". Similarly to the query, the content of the command is specified using multiple fields. This approach offers a very intuitive way of describing the application while relieving the control algorithm designer of the burden of dealing with the physical network implementation.

A token has nine fields, and its structure is:

$$\text{Token} = (q, c, n, a, v, T_i, T_f, L, Q),$$

where:

- $q \in \{0, 1\}$ specifies if it is a query or a command;
- $c \in \{0, 1\}$ specifies if it is a request or a response;
- n is the function to return for a query or the need for an acknowledgment for a command;
- a is the attribute of the query or the type of actuation;
- v is the required sampling rate (for a query) or the intensity of the actuation (for a command);
- T_i, T_f, are respectively the beginning and the end of the scope of the query [7] (i.e. "Give me humidity data from time T_i to time T_f");
- L [sec] is the latency requirement;
- Q is the quality of service requirement (bit error rate).

There are three types of actors: Virtual Controller (VC), Virtual Sensor (VS), and Virtual Actuator (VA).

A *Virtual Controller (VC)* contains the description of the control algorithm for the given application. If the application has more than one independent control algorithm, multiple Virtual Controllers have to be specified. In our case study, we have a single VC with an algorithm that needs information on both temperature and vibrations to take its decisions. The VC is only an abstraction of the control capabilities required by the application. This abstraction does not restrict our design space to a centralized control solution. In fact, in the physical implementation, the control algorithm described in a single VC could be implemented in a distributed fashion whenever it is convenient. Similarly, the functionalities of different Virtual Controllers could be implemented in the same physical component. Usually, designers already have a good idea of where the physical controller, or controllers, can be placed. Consequently, they can embed this location information into the VC and limit the design space exploration. The internal structure of a VC is a cyclic control routine. Figure 6 contains the pseudo-code of the VC routine for the monitoring application of Fig. 4. The code within the `while(true)` loop is called the *control cycle*. The number of queries and commands that can be generated during a control cycle must be limited. Consequently, no while loop with a query or command inside is allowed within a control cycle. This constraint is meant to avoid control cycles of infinite duration. Anyway, this does not result in a real restriction in the expressiveness offered to the algorithm designer because in most of the WSN applications that we analyzed, the control cycle can be expressed with less than about 100 queries or commands. Furthermore, the user can specify the time scope of the control cycle (how much time between two consecutive executions). If such parameter is not specified, we assume that the time scope is given by the the lowest T_i and the highest T_f of the generated queries or commands.

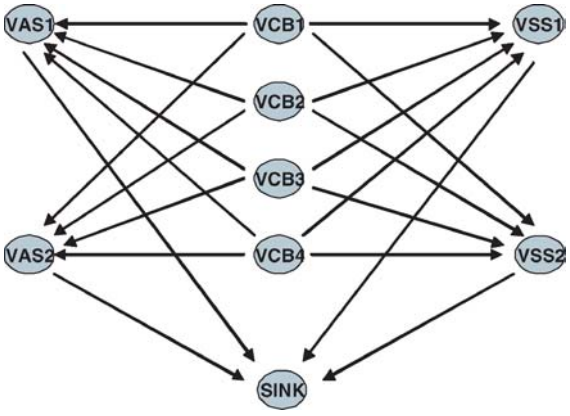

Fig. 5 Example of RialtoNet

A *Virtual Sensor (VS)* represents a sensing area. This abstraction is useful because designers know which are the areas that need to be sensed (and can embed this notion in the CRS), but they generally don't know how many sensors must be placed to cover that area and how they have to placed. This abstraction offers the possibility to embed in the application description a loose notion of topology that can be exploited further down in the design flow when MAC and routing algorithmshave to be tuned.

Similarly to the VC, there is not necessarily a one-to-one relationship between virtual sensors and physical sensors. The number and the type of physical sensors that will be used to implement a virtual sensor is an implementation choice.

A *Virtual Actuator (VA)* represents an actuation capability. Similarly to the VS, the user describes the position of the VA,

```
//Virtual Controller

VC
Connections:
VS1,VS2
VA1,VA2

While(true){
        //Query for vibration
        Q1=new query(1,0,avg,vib,5,init,t1,5s,e-3);
        //Query for temperature
        Q2=new query(1,0,avg,temp,10,t1,t2,10s,e-3);
  //Query for vibration with higher sampling rate
  Q3=new query(1,0,avg,vib,10,t2,t2+10,1s,e-3);
  //Query for vibration with lower samplin rate
        //to be performed only in non critical situation !!
  Q4=new query(1,0,vib,temp,1,t2,t2+15,5s,e-3);
        //Turn off the robot
  C1 = new command(0,0,x,turn_off,t2,t2,5s,e-3);

        send(Q1,VS"i");
        send(Q2,VS"i");

        await(VS1){
                response1=receive(VS1);
    data1=response1->data;
                attribute1=response1->attribute;
                if (((attribute==vib)&&(data1>threshold_vib))||
                        ((attribute==temp)&&(data1>threshold_num))){
                   send(C1,VA1);
                        send(C1,VA2);
                        send(Q3,VS1);
                }
                else{
                        send(Q4,VS1);
                }
        }

        await(VS2){
                response2=receive(VS2);
    data2=response2->data;
                attribute2=response2->attribute;
                if (((attribute2==vib)&&(data2>threshold_vib))||
                        ((attribute2==temp)&&(data2>threshold_num))){
                   send(C1,VA1);
                        send(C1,VA2);
                        send(Q3,VS2);
                }
                else {
                        send(Q4,VS2);
                }
        }
```

Fig. 6 VC for the case study

but the number and type of physical actuators that will be selected to implement its functionality is an implementation choice.

VA and VS are sequential threads of computation. They read the queries/commands at their inputs, perform their sensing/actuating task to satisfy those requests, and return data (if necessary) to the controller that sent the query/command. They are composed of two main functions: "Evaluate Inputs" and "Task". The "Evaluate Inputs" function specifies how the read semantics of the actor, while the "Task" function specifies how the actor fulfills the required sensing/actuation task.

Actors communicate through bidirectional, lossless, unbounded FIFO channels. Each channel is characterized by two separated queues, one for each direction. Connections

```
//Virtual Sensor

VS
Connections: VC;

While (true){

        Evaluate Inputs{
                if (input ==null)
                        Task();
                else {
                        Read Input();
                        Update parameters();
                        Task();
                }
        }
```

Fig. 7 VS and VA for the case study

are allowed only between a VC and a VA, and between a VC and a VS.

The user is free to specify any type of read and write semantics in the actors. For example, in our case study, the VC performs a blocking read at each of its inputs (`await` statement) before executing a corresponding atomic critical section. Conversely, the Virtual Sensors and Virtual Actuators of our example, perform a sensing task in a non-blocking read fashion. A pseudo-code for the Virtual Actuators and Virtual Sensors of our example is given in Fig. 7 (since in this case the VA and the VS have the same non blocking semantics, we report only the code for the VS).

Because of the large variety of applications that could be implemented using a WSN, it is very difficult to propose a single model of computation (MoC) that is able to offer the right level of expressiveness and yet to set the ground for developing automatic synthesis algorithms. Furthermore, the capabilities of the sensing and communication infrastructure are not related to the read and write semantics of the application. For example, the requirement that the link between two components should allow for a maximum latency of L seconds or that sensing should be performed at the rate of X [samples/sec], is a consequence of the content of the query and does not depend on the model of computation with which the application is specified. With this in mind, we think that the right approach is to allow designers to specify the selected read and write semantics, while the communication and sensing infrastructure should be derived independently.

4.2. Executing a rialto model: the RialtoNet

After the application is described, the description is translated into an internal representation called RialtoNet. Since we want to generate a set of requirements to design a sensing and communication infrastructure that is able to satisfy every possible request of the controlling algorithms, we need to evaluate all the various combinations of requests that

Virtual Controllers could generate. The RialtoNet is created precisely for an explicit exploration of all the possible combinations of queries and commands in a given application.

By analyzing the software code of every VC, we detect all the possible combinations of conditional statements involving a request, and for each of them we create a new independent component, called *VC Branch* (VCB). Each Virtual Sensor is modified into a *Virtual Sensor Skeleton* (VSS) and each Virtual Actuator into a *Virtual Actuator Skeleton* (VAS) that are obtained from the original code modifying the read and write semantics.

4.2.1. Generating the RialtoNet

We consider the case of analyzing the conditional branches in a single control cycle of the VC. For each conditional branch that involves the possibility of sending a request, we consider both scenarios: the one in which the branch is taken, and the one in which the branch is not taken. This analysis generates all the possible combinations of queries and commands within a single cycle. Each of these combinations generates a VCB. A VCB is composed by a sequence of "SEND" instructions that represent a possible combination. Consequently, the VCB is an actor that is only able to send a predetermined sequence of tokens ("source" actor). Since in the control cycle of the original VC code there is only a limited amount of queries and commands, also the number of "SEND" instructions in a VCB is limited. Since in the example of Fig. 6 there are two "if" statements, four VCBranches are generated (see Fig. 8).

Notice that a VCB does not contain the informations on read and write semantics specified in the original VC code.

```
VCB1
Connections:
VSS1,VSS2
VAS1,VAS2

  send(Q1,VSS1);
  send(Q1,VSS2);
  send(Q2,VSS1);
  send(Q2,VSS2);
  send(C1,VAS1);
  send(C1,VAS2);
  send(Q3,VSS1);
  send(Q3,VSS2);
```

```
VCB3
Connections:
VSS1,VSS2
VAS1,VAS2

  send(Q1,VSS1);
  send(Q1,VSS2);
  send(Q2,VSS1);
  send(Q2,VSS2);
  send(Q4,VSS1);
  send(C1,VAS1);
  send(C1,VAS2);
  send(Q3,VSS2);
```

```
VCB2
Connections:
VSS1,VSS2
VAS1,VAS2

  send(Q1,VSS1);
  send(Q1,VSS2);
  send(Q2,VSS1);
  send(Q2,VSS2);
  send(C1,VAS1);
  send(C1,VAS2);
  send(Q3,VSS1);
  send(Q4,VSS2);
```

```
VCB4
Connections:
VSS1,VSS2
VAS1,VAS2

  send(Q1,VSS1);
  send(Q1,VSS2);
  send(Q2,VSS1);
  send(Q2,VSS2);
  send(Q4,VSS1);
  send(Q4,VSS2);
```

Fig. 8 Virtual controller branches after branch separation

This is in line with our approach of considering only the requirements on the sensing and communication infrastructure that the WSN must support.

The VSS and VAS are sequential threads of computation. Similarly to the VCB, the VSS and VAS do not inherit from their originating actor the information regarding read and write semantics. They are composed by a "Task" function that is fired whenever their firing rules are satisfied. The "Task" code is inherited from their generating VS or VA. The firing rules are explained in the next section. VSS and VAS have an internal variable called *Progression Tag*. As we will show in the next section, this variable indicates the end of the time scope of the last query or command that has been served.

Every time a RialtoNet is generated, an extra actor called "Sink" is created. The Sink has only input channels and, as discussed in the next section, it is used to store the results of a RialtoNet execution.

Actors in a RialtoNet communicate also through unbounded, unidirectional, lossless, FIFO channels. Each VCB inherits the connections of its generating VC in the Rialto Model. The direction of these connections is from the VCB to the VSS or VAS. Each VSS and each VAS has an output connection to the Sink. The RialtoNet for the case study is shown in Fig. 5

4.2.2. Executing the RialtoNet:

In this Section we present the model of computation that regulates the execution of the RialtoNet. Before describing the read and write semantics, we need to introduce the END Token, a particular token that is automatically produced in the following two cases:

(1) From a VCB to all its output channels upon termination of its sequence of "SEND" instructions (see Fig. 8)
(2) From a VSS or VAS to the Sink whenever the its execution is terminated.

Its structure can be interpreted as:

$$\text{END} = (q, 0, 0, 0, 0, \textit{null}, \infty, \textit{null}, \textit{null})$$

The VCB follows a non-blocking write semantics. Since it is a source actor, no reading semantics needs to be specified. The Sink has a non-blocking read semantics. The VSS and VAS have blocking read and non-blocking write semantics. Since the blocking read rules for VSS and VAS are the same, we explain them only for the case of the VSS.

(1) The VSS stalls its execution until all its input queues have at least one token.
(2) Once that all the input queues are non empty, the VSS evaluates the first token of each of the input queues.

(3) If a VSS has END tokens in all its input queues, it sends an END token to the Sink and stops executing.
(4) Otherwise, the VSS selects the token with lowest T_f. If more than one token happens to have the same T_f and it is the lowest, all of these tokens are selected. Consequently, an END token is never consumed because it has ∞ in its T_f field.
(5) The VSS fires its sensing task. The output of the sensing task depends on the "a" and "v" fields of all the input tokens whose T_i field is less than or equal to the T_f field of the selected token. For example, in our case study VSS has four input queues and the first tokens at those queues are:
Input1:END
Input2:END
Input3:Q3 = (1,0, avg, vibration, 1000, $t2$, $t2$ + 10s, 1s, e − 3)
Input4:Q4 = (1, 0, avg, temperature, 10, t2, t2 + 15s, 5s, e − 3)
The VSS selects Q3 because it has lowest T_f and it advances its task until $t2 + 10s$. Requirements are generated for the interval ($t2$,$t2 + 10s$) such as:

(5) Sensing: vibrations at a rate of 1000 samples/sec and temperature at a rate of 10 samples/sec.
- Communication: latency of 1s (the most restrictive among the two) and message error rate of 10^{-3}.

This is the set of requirements that the VS must satisfy in order to serve all the possible queries within that time scope.

The last action of a firing is the generation of a *requirement token*. This token has the same nine fields of the other tokens, but instead of abstracting a query or a command, it embeds information of the generated requirements. In our example, the VSS generates a Requirement Token: Out = (1, 0, [avg, avg], [vib, temp], [1000, 10], $t2$, $t2$ + 10s, 1s, e − 3).
This token encodes the following requirements: "From time $t2$ to time $t2 + 10$ seconds, the VS must be able to sense vibration and temperature at a rate of respectively 1000 sam/sec and 10 sam/sec., and return the average. Furthermore, it must be able to communicate with the Virtual Controller with a latency of 1 sec and a message error rate of 10^{-3}".
The VSS sends the Requirement Token to the Sink. Since the Sink receives only Requirement tokens (and END tokens which have a fixed structure), and it is the only one receiving them, there is no need to distinguish this token from the other types.

(6) The selected token (in our case Q3) is consumed, meaning that it is removed from its input queue and destroyed.

The Progression Tag update happens at every firing of the sensing/actuating task. At the end of the firing, the progression tag of the VS/VA is set to the value of the T_f field of the selected token.

Queries sent in the same VCB connection must have non overlapping time scopes. This is to avoid the situation in which, after advancing to serve a query, a VS would have to backtrack to serve another query with different requirements. Once the code for a VCB is generated, this condition can be easily checked. If such a case is detected, the VCB code is modified and every couple of overlapping queries is replaced by three non overlapping queries. Queries emitted from the same VC branch must have non decreasing T_f field. This is to avoid the phenomenon of "sending a query to the past".

The execution terminates when each VSS and each VAS has sent an END token to the Sink.

4.2.3. *Determinism*

The RialtoNet semantics is based on a deterministic MoC: there is only one possible behavior for the input and output sequences of the actors.

Let T denote the set of all the finite and infinite sequence of tokens, including the empty sequence ($\perp$). Consider the prefix order ($\leq$) such that $s_1, s_2 \in T$, $s_1 \leq s_2$ if for all the $n \in N$ for which $s_1(n)$ is defined, $s_1(n) = s_2(n)$. For instance, if $s_1 = (a, b, c, d)$ and $s_2 = (a, b, c, d, e, f)$, then $s_1 \leq s_2$. A simple extension of the prefix order is the pointwise prefix order ($\sqsubseteq$). Assume (a_1, a_2),$(b_1, b_2) \in (T \times T)$, the pointwise prefix order is defined as: $(a_1, a_2) \sqsubseteq (b_1, b_2)$ if $a_1 \leq b_1$ and $a_2 \leq b_2$. The set T^N with the pointwise prefix order $(T^N, \sqsubseteq)$ is a complete partial order (CPO) [32]. A function F is monotonic with respect to $(T^N, \sqsubseteq)$ if for $a, a' \in T^N$ and $a \sqsubseteq a'$, it follows $F(a) \sqsubseteq F(a')$.

Similarly to the processes of a Kahn Process Networks (KPN) [29, 30]. the VAS and VSS are monotonic. Monotonicity is a property inherited from the blocking read mechanism and by the fact that the choice of the token to be consumed is deterministic and based on the T_f field and not on the order of arrival of the candidate tokens. Furthermore, since the input sequences are bounded by the total number of queries/commands declared in the VCB, the VAS and VSS are also continuous with respect to $(T^N, \sqsubseteq)$. The VCB and the Sink are trivially continuous functions since they are source and sink function. Since a RialtoNet is composed of continuous functions under a complete partial order, it converges to a least fixed point [29]. Consequently, the least

fixed point is the only behavior and the model is deterministic [32].

4.2.4. Deadlock free

Another important property of the RialtoNet execution is that it does not deadlock. Deadlock may happen only if a VSS or VAS waits in vain for a token that will never arrive. The introduction of the END token is tailored to avoid this problem. The idea of introducing the END query to resolve unwanted deadlocks can be seen as a particular case of the "null" message introduced by Misra in [33] when dealing with asynchronous parallel simulations.

4.2.5. Conservative advancement

The proposed blocking read mechanism forces the VSS and VAS to have a *conservative advancement* behavior. This means that before firing their sensing/actuating task, they need to wait for all their input queues to have a token, and when they advance they do it only up to the lowest T_f. In this way we can capture all possible evolutions in the behavior of an application without being forced to store subsequent stages of it for the purposed of rollback.

Consider what could happen in the interaction between VCB1,VCB3 and VS1 in our case study. Figure 9 illustrate a scenario where these actors exchange two particular sequences of queries (the definition of the queries is given in Fig. 6). Assume that the VS1 does not perform the proposed blocking read. In this case VS1 would serve Q4 from VCB3 and advance its task to the Progression Tag value $t2 + 15s$ before evaluating Q3 from VCB1. As a result, since $Q3$ has higher sampling rate requirement than $Q4$, in order to correctly serve $Q3$, VS1 would have to go back to $t2$, void its latest execution and serve $Q3$. This simple example shows that the blocking read mechanism is a clean way to capture all the scenarios without the need of supporting rollback. The correctness of the specification capturing is a consequence of the determinism of the proposed MoC.

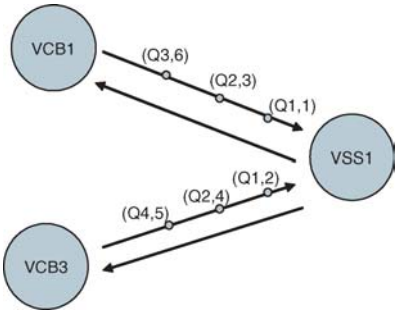

Fig. 9 Example of conservative advancement

4.2.6. RialtoNet vs. Rialto model

The END query and the blocking read mechanism are only used to ensure the determinism of the execution of the RialtoNet. They are not specific to the application described in the Rialto Model and they need not be implemented in the final system implementation. On the other hand, they make it possible to derive precise requirements on the performance properties of the communication links as well as the number and type of sensors that have to be used. In other words, RialtoNet is an efficient model to gather sufficient information for the design of the sensing and communication architecture because the characteristics of the computation of the control algorithms are abstracted away.

4.3. Requirement generation

Once the program has terminated, the Sink actor contains the evolution of the sensing modalities, latency and bit error rate requirements over the time scopes of all the Virtual Sensors and Virtual Actuator. This information is captured as a set of traces from which we derive the requirements that the communication and sensing infrastructure must satisfy to ensure correct functionality of the application.

For the subsystem of Fig. 4 with only two robots, the execution of the Rialto model produces the traces of Fig. 10. Consequently, to satisfy the application requirements, we need to place within a VS area a number of sensors that is sufficient to sample vibrations at 10 samples per second and temperature at 10 samples per second. Furthermore, the communication infrastructure needs to be able to report data to the controller with a latency requirement of 5 seconds with a packet error rate of 10^{-3}.

For the overall system of five robots illustrated in Figure 3, the design requirements are [35]: an end-to-end delay of 10 seconds, a packet error rate around the 2 σ distribution, and a sampling rate of 10 samples per second for both temperature and vibration. The information on the po-

```
Control Cycle Time Scope= t2+15sec;

Sensing:

VS1:
 temp:
  (t1,t2)-> 10 sam/sec;
 vib:
  (init,t1)-> 5 sam/sec;
  (t2,t2+10)-> 10 sam/sec;
  (t2+10,t2+15)-> 1 sam/sec;

VS2:
 temp:
  (t1,t2)-> 10 sam/sec;
 vib:
  (init,t1)-> 5 sam/sec;
  (t2,t2+10)-> 10 sam/sec;
  (t2+10,t2+15)-> 1 sam/sec;

Communication:

 VS1-VC:
  (init,t1)->(5sec,e-3);
  (t1,t2)->(10sec,e-3);
  (t2,t2+10)->(1sec,e-3);
  (t2+10,t2+15)->(5sec,e-3);

 VS2-VC:
  (init,t1)->(5sec,e-3);
  (t1,t2)->(10sec,e-3);
  (t2,t2+10)->(1sec,e-3);
  (t2+10,t2+15)->(5sec,e-3);

 VC-VA1:
  (t2,cycle)->(5sec.,e-3);

 VC-VA2:
  (t2,cycle)->(5sec.,e-3);
```

Fig. 10 Requirements for the case study

sition of the sensing areas is also used to create a first "loose notion" of WSN topology. Specifically, each virtual sensor represents a cluster of sensors in the final implementation. take this cluster topology, substitute the clusters with an adequate number of nodes per cluster and create a more refined topology.

5. Bridging SNAPP and SNIP

In this section, we present two protocols that are currently part of our framework and their interface to the application.

An important and usually non trivial step in the top-down refinement process associated with the move from one layer of abstraction to the next consists in analyzing application requirements on the end-to-end (E2E) delay and translating them into a hop-to-hop (H2H) delay which is simpler to handle and of direct impact to the protocol design. The ability of performing this refinement is subject to the capability of characterizing the interaction among the different layers of the protocol solution using a mathematical framework. The mathematical framework allows us to capture the requirements of the design functionality and performance as a constrained optimization problem. The solution to this problem provides the parameters to derive the final protocol implementation. Once the trade-off equations are derived and solved as an optimization problem, all the protocol parameters are automatically synthesized. The formalism and the capability of offering end-to-end guarantees instead of local guarantees is what distinguish our approach from the previous protocol design for WSNs.

The use of parameterized protocols allows us to effectively restrict the large design space to a few parameters. In addition, since the protocols are developed with a specific mathematical model in mind, we can easily gouge the effects of changing these parameters on the overall network performance. This predictive ability prevents the need for extensive simulation and allows for the ease of comparison with other protocols.

Furthermore, the ability to obtain quick performance estimates is vital for conducting an effective design space exploration. Since the protocol itself is constructed by the designer, our approach does not discount the value of designer intuition. We believe that a system-level design technique which depends entirely upon automated synthesis and does not allow the designer any "hooks" into the design process will not be successful. Designer experience is invaluable in domains such as WSNs that push the limits of current technology. However, when we couple designer intuition with mathematical models, we can gain a deeper understanding of system behavior.

With the continued scaling of Moore' s Law and advances in MEMS fabrication, the cost of individual nodes is expected to drop in the years to come. This will allow high density WSN deployments. Since in most applications a long network lifetime is required and energy constraints are usually tight, high density WSNs are attractive. A high node density allows the WSN to turn off multiple nodes while still remaining operational. On the other hand, designing protocols for such a scenario becomes an interesting challenge due to the collision problems that arise from having a high density of nodes with a limited number of communication channels.

With this in mind we set out to design the Routing and MAC layers of the protocol stack that leverages high node density to offer robustness and optimize for energy consumption. Although these protocol solutions have different characteristics to accommodate different scenarios, they are all characterized by a mathematical model that exposes their guarantees on end-to-end delay and packet loss. The decision of which of these protocols is to be chosen depends on the topology and the characteristics of the specific application. For example, in case of a cyclic control routine, as in our application domain, we developed two protocols, one for the case of homogeneous layout topologies where nodes are uniformly placed at random in a small space, and another that is tailored to clustered topologies as in our case study.

In the next subsections we give a brief description of these protocols, and we show how the synthesis of the protocol parameters is performed for our industrial example.

5.1. Randomized protocol for homogeneous topologies

In [24], we presented an example of a randomized protocol designed according to this methodology. The proposed protocol is based on the joint optimization of a randomized routing protocol, a MAC protocol, and a duty cycling one that allow for performance and reliability while leveraging node density. The solution offers the following services:

(1) End-to-End (E2E) delay guarantee. The two sigma distribution of the E2E packet delay between a node and the controller must stay within τ seconds: $P[E2E \leq \tau] \geq 0.96$.
(2) Error Rate guarantee Each packet must reach the destination probability at least Ω: $P[correct] \geq \Omega$.

The behavior of a node can be explained considering the state machine of Figure 11.

- *SLEEP STATE*: the node turn off its radio and starts a grenade timer whose duration is an exponentially distributed random variable of intensity μ. When the timer expires, the node goes to the WAKE UP state.

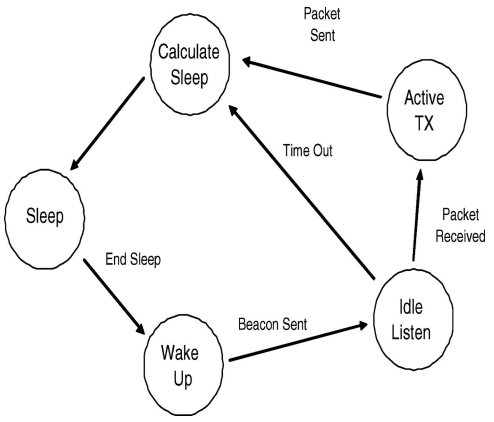

Fig. 11 Randomized protocol

- *WAKE UP STATE*: the node turn its radio on and broadcasts a message indicating its location and that it is ready to receive (Beacon message). The node goes to the IDLE LISTEN state.
- *IDLE LISTEN STATE*: the node starts a grenade timer of a fixed duration that must be long enough to completely receive a packet. If a packet is received, the timer is discarded and the node goes to the ACTIVE TX state. Otherwise if the timer expires before any packet is received, the node goes to the CALCULATE state.
- *ACTIVE TX STATE*: the node calculates the size of the forwarding region (FwR). The FwR is the region between the maximum and minimum distance ($d_{\max}, d_{\min}$) at which the next hop must be. The node waits for the first beacon coming from a node within the FwR and forwards the packet to it. After the transmission is completed it goes to the CALCULATE state.
- *CALCULATE STATE*: the node calculates the intensity parameter μ for the next sleeping time and generates an exponentially distributed random variable of mean $1/\mu$. After this the node goes back to the SLEEP state.

Consequently:

(1) the selection of the next hop is a random choice among nodes of a calculated region
(2) the duty-cycling algorithm is randomized
(3) the MAC is random based and does not implement any acknowledgment and retransmission scheme
(4) the working point of each node is determined by the size of the FwR and the wake up intensity μ.

These parameters can be adaptively tuned in order to satisfy delay and error rate constraints as well as to optimize for power consumption [24]. We further introduce an initialization and network configuration protocol that allow for plug and play of the described solution.

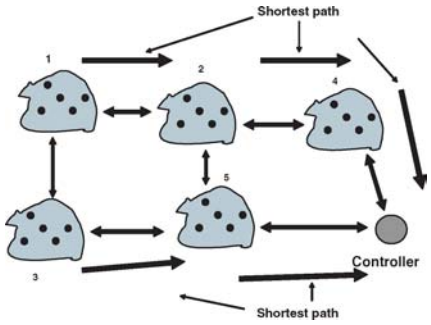

Fig. 12 Connectivity graph

5.2. SEmi-RANdom protocol for clustered topologies: SERAN

For naturally clustered environments, as in our example, we developed a semi-random protocol stack called SERAN, which covers two layers of a classical protocol stack: routing and MAC. In this section, we present a brief overview of SERAN, while we refer to [35] for a more detailed description and performance analysis. The case study of Fig. 3 represent a naturally clustered topology, where nodes are deployed in clusters whose positions are known a priori.

5.2.1. Routing Algorithm

Routing over an unpredictable environment is notoriously a hard task. High node density makes the problem easier to solve. The idea is to have a set of nodes within transmission range that could be candidate receivers; at least one of them will offer a good link anytime a transmission is needed.

The routing solution of SERAN is based on a semi-random scheme that reduces the overhead of purely random approaches. In SERAN, the sender has knowledge of the region to which the packet will be forwarded, but the actual choice of forwarding node is made at random. This approach is motivated by the fact that the clustered topology of the sensor network for robots monitoring in a manufacturing cell is known a priori.

A connective graph like the one reported in Fig. 12 can be derived from the given cluster topology [35]. In the graph, an arc between two clusters means that the nodes of the two clusters are within transmission range. We further assume that the nodes share the same communication channel. Then, the first step of the SERAN routing algorithm consists of calculating the shortest path from every cluster to the PLC and generating the minimum spanning tree as shown in Fig. 12.

Assuming that a particular node in Cluster 1 must forward a packet to the PLC, the proposed routing algorithm works as follows:

(1) the node that has the packet selects randomly a node in Cluster 2 and forwards the packets to it;
(2) the chosen node determines its next hop by choosing a node randomly in Cluster 4, and so on;

In other words, packets are forwarded to a randomly chosen node within the next-hop cluster in the minimum spanning tree leading to the PLC.

5.2.2. *Hybrid MAC*

The first priority for the design of our MAC is ensuring robustness against topology changes. Since nodes failure is a common phenomenon for WSN, we design a MAC that is able to support the addition of new nodes for preserving the high level of density required to ensure robustness. This flexibility is usually obtained by using random based access schemes. In the WSN domain, an interesting example of this idea is presented in BMAC [37].

High density unfortunately introduces a large number of collisions. This drawback becomes crucial in our case because we have only one channel that can be used for communication. To reduce collisions, a deterministic MAC is used. A well-known deterministic approach is SMAC [36], where the network is organized according to a clustered TDMA scheme. Our MAC solution is based on a two-level semi-random communication scheme. This offers robustness to topology changes and node failures that is typical of a random based MAC protocol and robustness to collision that is typical of a deterministic MAC protocol

High Level MAC: The higher level regulates channel access among clusters. A weighted TDMA scheme is used such that at any point in time, only one cluster is transmitting and only one cluster is receiving. During a TDMA cycle, each cluster is allowed to transmit for a number of TDMA-slots that is proportional to the amount of traffic it has to forward. The introduction of this high level TDMA structure has the goal of limiting interference between nodes transmitting from different clusters. The time granularity of this level is the TDMA-slot (see Fig. 13). After the two clusters terminated their transmitting TDMA-slot, another TDMA-slot (called the *actuation slot*) is reserved for the PLC. During this slot, the PLC sends a message to the actuator of each robot to continue operating or to switch the robot off.

Low Level MAC: The lower level regulates the communication between the nodes of the transmitting cluster and the nodes of the receiving cluster within a single TDMA-slot. It has to support the semi-random routing protocol presented in V-B.1, and it has to offer flexibility for the introduction of new nodes. This flexibility is obtained by having the transmitting nodes access the channel in a p-persistent CSMA fashion [38]. The random selection of the receiving node is obtained by multi-casting the packet over all the nodes

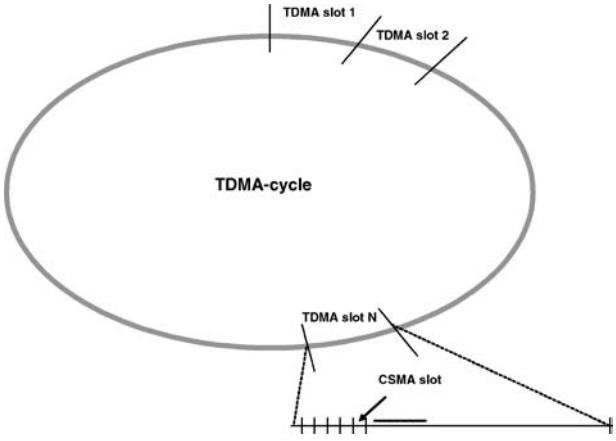

Fig. 13 TDMA-Cycle representation

of the receiving cluster, and by having the receiving nodes implement a random acknowledgment contention scheme to prevent duplication of the packets. Calling CSMA-slot the time granularity of this level (see Fig. 13), the protocol can be summarized as follows:

(1) Each of the nodes of the transmitting cluster that has a packet tries to multi-cast the packet to the nodes of the receiving cluster at the first CSMA-slot with probability p.
(2) At the receiving cluster, if a node receives more than one packet, it detects a collision and discards all of them. If it has successfully received a single packet, it starts a back-off time T_{ack} before transmitting an acknowledgment. The back-off time T_{ack} is a random variable uniformly distributed between 0 and a maximum value called T_{ackmax}. If in the interval between 0 and T_{ack}, it hears an acknowledgment coming from another node of the same cluster, the node discards the packet and does not send the acknowledgment. In case of a collision between two or more acknowledgments, the involved nodes repeat the back-off procedure. At the end of the CSMA-slot, if the contention is not resolved, all the receiving nodes discard the packet.
(3) At the transmitting node side, if no acknowledgment is received (or if only colliding acknowledgments are detected), the node assumes the packet transmission was not successful and it multi-casts the packet at the next CSMA-slot again with probability p. The procedure is repeated until transmission succeeds.

In this approach, nodes need to be aware only of the next-hop cluster connectivity and do not need a neighbor list of next hop nodes. We believe this is a great benefit because cluster based connectivity is very stable, while neighbor lists of nodes are usually time-varying (nodes may run out of power and other nodes may be added) and their management requires significant overhead.

In [24], it is shown how a similar acknowledgment contention scheme reduces significantly the packet duplication effect. However, we still cannot guarantee that duplicate packets are not generated. This may happen if a receiving node does not hear the acknowledgment sent by another node in the same cluster. Although these duplicate packets are detected at the PLC, they still create an extra amount of traffic in the network. In [35], we show how protocol parameters can be optimized to reduce further this traffic overhead.

Energy Minimization: In most of the proposed MAC algorithms for WSNs, nodes are turned off to save energy whenever their presence is not essential for the network to be operational. Similar to this approach, our duty-cycling algorithm leverages the MAC properties and does not require extra communication among nodes. During an entire TDMA cycle, a node has to be awake only when it is in its listening TDMA-slot or when it has a packet to send and it is in its transmitting TDMA-slot. For the remainder of the TDMA cycle, the node radio can be turned off.

5.2.3. *Protocol parameters synthesis*

The working point of the communication protocol is determined by tuning a set of parameters such as the TDMA schedule, the duration of the TDMA-slot, and the channel access probability p.

TDMA Schedule: Referring to Fig. 12, assume for now that the average generated traffic at each cluster is the same. According to our shortest cluster path routing solution, packets are transferred cluster-by-cluster along the shortest path until they reach the PLC. Consequently, clusters close to the PLC, have a higher traffic load since they need to forward packets generated within the cluster as well as packets coming from upstream clusters. In the example of Fig. 12, the average traffic intensity that Cluster 4 experiences is three times the traffic intensity experienced by Cluster 1. Consequently, we can assign one transmitting TDMA-slot per TDMA-cycle to Cluster 1, two transmitting TDMA-slots to Cluster 2 and three transmitting TDMA-slot to Cluster 4. Similarly, on the other path, the number of associated TDMA-slots per cluster can be assigned. Therefore, assuming we have P paths and calling B_{i} the number of clusters in the $i-th$ path, we have a total of $\sum_{i=1}^{P} Bi(B_i+1)/2$ TDMA-slots per TDMA-cycle.

Once we decide the number of TDMA-slots per TDMA-cycle for each cluster, we need to decide the scheduling policy for transmitting and receiving. We consider an interleaved schedule (Fig. 14). For each path, the first cluster to transmit is the closest to the PLC (Cluster 4). Then Cluster 2 and Cluster 4 again. Then Cluster 1, 2 and 4, and similarly on the other path. This scheduling is based on the idea that evacuating the clusters closer to the PLC first, we minimize the storage requirement throughout the network.

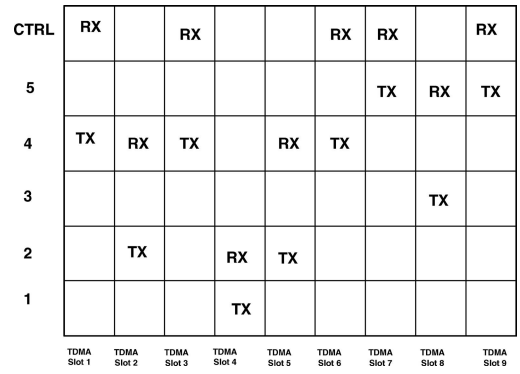

Fig. 14 Scheduling: clusters close to the Controller are evacuated first

TDMA-slot duration: Starting from the latency requirements outlined in the previous section, we want to determine the duration of a TDMA-slot such that the latency requirements are satisfied and energy consumption minimized.

Call $D_{\max}$ the latency requirement, P the number of paths, B_{i} the number of clusters per path, and S the duration of the TDMA-slot. The requirement on S is [35]:

$$S \leq S_{\max} = \frac{D_{\max}}{\max_{1,\ldots,P} B_i + \sum_{i=1}^{P} B_i(B_i+1)/2} \tag{1}$$

Since, as shown inn [35], the total energy consumption is a monotonically decreasing function of S, the optimal TDMA-slot duration is $S = S_{\max}$. Using the number of our case study, this means a TDMA-slot duration of $S_{\mathrm{opt}} = 1120ms$.

Random access parameter: The random access parameter p needs to be set such that all the nodes in the cluster are able to forward their packets during a TDMA-slot. In [35], we model the packet transmission process as a Discrete Time Markov Chain [31]. Let τ_{k} denote the expected number of CSMA-slots that are needed to forward all the packets in the cluster given that there are k packets at the beginning of the TDMA-slot. Then, we have [35]:

$$\tau_k = \sum_{j=1}^{k} \frac{1}{cpj(1-p)^{j-1}} \tag{2}$$

The sum in (2) is a convex function of p, hence for different values of k there is an optimal value of p (that we refer to as p_k) that minimizes τ_k. In our case the optimal parameter is $p_{\mathrm{opt}} = 0.1348$. Furthermore, in [35] we show how this choice ensures a packet error rate below 1%. With these parameters the communication protocols satisfies the given constraints and reduces the energy consumption of almost an order of magnitude when compared to the most common protocol solutions for WSNs. Since k depends on the sensing

rate requirements that we get from Rialto and the duration of the TDMA-slot, we are now able to set all the parameters to the optimal working point.

6. Mapping and implementation

After creating the network infrastructure, the final step of the design flow consists in mapping the controlling algorithm onto the controller hardware platform, and mapping the communication protocol onto the wireless nodes.

In general, mapping functionality over a WSN is a difficult process. The main reason is that if the communication infrastructure is not created based on a clear mathematical model, and a clear set of guarantees on latency and packet error rate, the verification of the final solution is difficult and often replaced by a validation with time consuming simulations. Furthermore, whenever the simulations do not show the intended behavior, the design cycle has to go back to the protocol design stage.

With our approach, most of this hassle is avoided and that is because the network architecture generated using Rialto and the synthesis of the protocol parameters of the SNAPP instance is guaranteed by construction to be able to support the application while being energy efficient. Consequently, the mapping process is extremely simplified:

(1) The first step consists in mapping the controlling algorithm into the hardware platform of the PLC. This represents a classical embedded systems mapping problem (i.e. not specific of the WSN domain) and it can be performed with classical mapping tools. Consequently, we offer only some quick remarks on this step. Metropolis [27] is a design environment that was developed to support Platform Based Design. The advantage of using Metropolis in our design flow is that it supports any type of model of computation for the functional description. This property allows us to implementing our philosophy of leaving the user freedom to select the preferred model of computation for describing the control routines. Following the PBD, after the control routine is specified, an abstraction of the hardware platform of the PLC must be provided in order to drive the mapping process.
In many industrial practices, the PLC comes already with a software development environment. Although these environments do not offer the same flexibility and formal methodology of Metropolis, they are commonly used by plant designers because they are user friendly and sold by companies with good customer support. However, we believe that a more formal approach to this problem should be pursued since a bad mapping often leads to suboptimal or faulty implementations.

(2) The second step is to map the communication protocol on the physical nodes. Since the communication protocols of the SNAPP are already described in a distributed fashion, the parametrized code for each node can be easily developed using the software interface of the nodes. Most often, this interface is given by TinyOS and the parametrized code can be written using NesC.
The actual setting of the parameters of the nodes to determine their working point is obtained using an initialization algorithm that kicks in when the nodes and the PLC are switched on and allows for self-adaptation of the network to the optimal working conditions. Furthermore, to preserve the correct behavior of the communication infrastructure, network management algorithms are automatically run on the network on a periodical basis.

7. Related work

Standards: A number of standards for open communication in sensor networks have been proposed. The most well-known are the BACnet and LonWorks standards, developed for building automation [1]. These standards are geared toward well-defined application areas, and are built on top of fairly well defined network structures. Hence, many of the exciting new developments that are emerging from the wireless sensor network community cannot be accommodated within these frameworks. At the same time, the application-specific functionality of both BACnet and LonWorks can easily be overlaid on top of the service-based model proposed in this paper.

The approach that is closest to our approach is TinyDB [10]. While TinyDB is also based on the Query/Command paradigm, its main goal is to define the interface and an implementation of a specific service, the query service, rather than defining a universal service platform interface. Hence, TinyDB does not include several auxiliary services that are necessary in many sensor network applications.

There is a number of standards in the making at the ad-hoc wireless network layer. The best known is Zigbee, advocated by a consortium of companies [6]. ZigBee defines an open standard for low-power wireless networking of monitoring and control devices. It works in cooperation with the IEEE 802.15.4 standard, which focuses on the lower protocol layers (physical and MAC). Instead, ZigBee defines the upper layers of the protocol stack, from network to application, including application profiles. From our perspective, Zigbee represents only one possible way to realize a network. The services proposed in [7] can be easily deployed in and on top of a Zigbee realization or alternative implementations such as Bluetooth Scatternets.

The research community has proposed several implementations for each of the services offered by the Sensor Network Service Platform. Several attribute-based schemes that assign, modify, or translate names have already been proposed [11–13]. These schemes are different from the one used in the Internet, where nodes are usually addressed individually and are identified by a unique identifier called IP address. DNS (Domain Name Server [13]) holds an association between names and IP addresses and upon request provides a source node with the IP address of the destination host before message delivery (early binding). INS [11] proposes an overlay network of Intentional Naming Resolvers (INRs) that associates attribute-based names with IP addresses and binds them at message delivery time (late binding) rather than at request resolution time. In INS, naming is done using name-specifiers based on a set of attributes and their values. Attribute-based naming is also proposed in [12] where name matching, instead of being done by special-purpose network elements, is distributed across the network and names are associated with (attribute, value, operation) tuples, where operation specifies the type of operation to be used for name matching.

Networks of sensors (mostly wired) have further been used in automation and manufacturing. A number of standards have been developed within the IEEE to deal with different manufacturers.

More specifically, the IEEE 1451.2 [15] standardizes both the key sensors (and actuators) parameters and their interface with the units that read their measures (or set their values). In particular, the standard defines the physical interface between the Smart Transducer Interface Module (STIM), which includes one or more transducers, and the Transducer Electronic Data Sheet (TEDS) containing the list of their relevant parameters, and the Network Capable Application Processor (NCAP), which controls the access to the STIM. IEEE 1451 was defined to improve the reusability of the network and component solutions for sensor networks within manufacturing plants. Although the initial targets were wired networks, the applicability of its concepts appeals to a wireless solution. IEEE 1451 presents already the concept of logical components (e.g, a sensor identifies a group of sensing devices rather than a single hardware component). Nevertheless, the IEEE 1451 standards are specifically targeted to the design of interfaces and they can hardly be generalized to capture application characteristics.

Tools: Several tools to support the design of WSNs are available. The most common design methodology for WSNs starts with the description of the protocol specifications using the NesC/TinyOS stack [16]. The NesC/TinyOS platform, developed at U.C. Berkeley, leverages a "method call" model of computation. It was designed to describe component-based architectures using a simple event-based concurrency model. This platform has then been enriched with a simulation environment called TOSSIM [17]. Its success is also related to the wide spreading of the hardware platforms of the Mica family [18]. Remarkably, the combination of Mica and TinyOS allowed for the development of many WSN applications.

Alternatively, protocol solutions are simulated using environment such as OMNET + + [19] or VisualSense [20] and then implemented in NesC/TinyOS. Omnet + + is a discrete event simulator developed by Andras Varga at the Technical University of Budapest. Although not specifically targeting the WSN domain, Omnet + + is widely used within the communication community for protocol simulations.

Visualsense is a modeling framework for WSN developed as part of the Ptolemy project at U.C.Berkeley [21]. It is an extension of a discrete-event model with an extra capability of describing properties of the wireless connectivity. Visualsense is a powerful tool to model and evaluate protocol solutions under different scenarios. Although an effort to move to a higher layer of abstraction is visible, especially with Visualsense, the current design flows are too oriented toward a bottom-up approach.

An attempt of raising the level of abstraction is presented in [22], where a classification for node communication mechanisms is introduced to allow for a higher level description of the network algorithms. In [23], a design methodology is presented. That methodology is based on a bottom-up part for the description of network algorithms, a top-down part to describe the application, and a mapping process to deploy the software code onto the nodes. The overall method fits with the PBD paradigms advocated in this paper but leverages different layers of abstraction. Our approach emphasizes the control based nature of WSN applications and offers a clear semantics and set of primitives to interpret timing issues at a very high level, hence providing a clear level of abstraction for the application designer. In [18, 25, 26], a description of current efforts on hardware platforms is provided. Low power design for the WSN domain has become a very active research topic.

8. Conclusions

We presented a Platform-Based-Design methodology for wireless sensor networks. First we identified the appropriate layers of abstraction that forms the pillars of the methodology: an application interface that allow the user to describe the application independently from the network implementation (the Sensor Network Service Platform, SNSP), a platform that is composed of a set of power aware communication protocols (SNAPP), and an abstraction that captures the implementation architectures (the Sensor Network Implementation Platform, SNIP).

We presented an example of the proposed design flow in an application of industrial control using wireless sensor networks. We introduced Rialto, a framework that helps translating the application description obtained using the SNSP into requirements on the communication links and sensing infrastructure that are the interface to the SNAPP. Starting from these constraints and information on the system topology, an adequate communication protocol among the ones available in the SNAPP is selected. The working point of this protocol is subsequently decided as the result of solving a constrained optimization problem where the constraints are given by the requirements generated by Rialto and the cost function is obtained using an abstraction of the candidate hardware platform.

Our future work will consist in further populating the protocols of the SNAPP and in generalizing our mapping strategy to diversify the supported application space.

References

1. D. Snoonian, Smart buildings, IEEE Spectrum (Sept.2003) 18–23.
2. J. Rabaey, E. Arens, C. Federspiel, A. Gadgil, D. Messerschmitt, W. Nazaroff, K. Pister, S. Oren and P. Varaiya, Smart energy distribution and consumption information technology as an enabling force, white paper, http://citris.berkeley.edu /SmartEnergy/SmartEnergy.html.
3. G. Huang, Casting the wire, Technology Review July/August (2003) 50–56.
4. F. Boekhorst, Ambient intelligence: The next paradigm for consumer electronics, in: *Proceedings IEEE ISSCC 2002* (San Francisco, February 2002).
5. IEEE 802.15 WPAN Task Group 4 (TG4), http: //www.ieee802.org/15/pub/TG4.html
6. The Zigbee Alliance, http://www.zigbee.org
7. M. Sgroi, Adam Wolisz, Alberto Sangiovanni-Vincentelli and Jan M. Rabaey, A service-based universal application interface for ad-hoc wireless sensor networks whitepaper, U.C. Berkeley, (2004).
8. A. Sangiovanni-Vincentelli and A. Ferrari, System design—Traditional concepts and new paradigms, in: *Proceedings of ICCD 99*, Austin, (October 1999), pp. 2–12.
9. A.L. Sangiovanni-Vincentelli, L. Carloni, F. De Bernardinis and M. Sgroi, Benefits and challenges for platform-based design, in: *Proceedings of the Design Automation Conference (DAC'04)* San Diego, CA, USA, (June 2004).
10. S. Madden, The design and evaluation of a query processing architecture for sensor networks, Ph.D. Dissertation, UC Berkeley (2003).
11. W. Adije-Winoto, E. Schwartz, H. Balakrishnan and J. Lilley, The design and implementation of an intentional naming system, in: *Proceedings of Symposium on Operating Systems Principles* (Dec. 1999).
12. C. Intanagonwiwat, R. Govindan and D. Estrin, Directed diffusion: A scalable and robust communication paradigm for sensor networks, in: *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networks (MobiCom 2000)*, Boston, Massachusetts, (August 2000).
13. P. V. Mockapetris and K. Dunlap, Development of the Domain Name System, in: *Proceedings of SIGCOMM'88* (Stanford, CA, 1988).
14. D. Mills, Internet time synchronization: The network time protocol, in: *Global States and Times in Distributed Systems* IEEE Computer Society Press, (1994).
15. IEEE 1452.2, Standard for a smart transducer interface for sensors and actuators—transducer to microprocessor communication protocols and transducer electronic data sheet (TEDS) formats, IEEE (1997).
16. D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer and D. Culler, The nesC language: A holistic approach to networked embedded systems, in: *Proceedings of Programming Language Design and Implementation (PLDI) 2003* (June 2003).
17. P. Levis, N. Lee, M. Weksh and D. Culler, TOSSIM: Accurate and scalable simulation of entire TinyOS application, *SENSYS* 03 (2003).
18. J. Hill and D. Culler, Mica: A wireless platform for deeply embedded networks, IEEE Micro 22(6) (2002) 12–24.
19. A. Varga, The OMNeT++ discrete event simulation system, in: *European Simulation Multiconference* (June 2001).
20. P. Baldwin, S. Kohli, E.A. Lee, X. Liu and Y. Zhao, Visualsense: Visual modeling for wireless and sensor network systems, UCB ERL Memorandum UCB/ERL M04/8 (April 23, 2004).
21. http://ptolemy.eecs.berkeley.edu
22. Y. Yu, B. Hong and V.K. Prasanna, Communication models for algorithm design in wireless sensor networks, *IPDPS'05* (2005).
23. A. Bakshi and V.K. Prasanna, Algorithm design and synthesis for wireless sensor networks, *ICPP'04* (2004).
24. A. Bonivento, C. Fischione and A. Sangiovanni-Vincentelli, Randomized protocol stack for ubiquitous networks in indoor environment, in: *CCNC* (2006).
25. J. Rabaey et al., PicoRadios for wirless sensor networks: The next challenge in ultra-low-power design, in: *ISSCC 2002*, (Feb. 2002).
26. J. Kahn, R. Katz and K. Pister, Next century challenges: Mobile networking for smart dust, *MobiCom* (1999).
27. F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone and A. Sangiovanni-Vincentelli, Metropolis: An integrated electronic system design environment, [Journal Paper] Computer 36, (4) (2003) pp. 45–52 Publisher: IEEE Comput. Soc, USA.
28. J.R. Burch, R. Passerone and A.L. Sangiovanni-Vincentelli, Using multiple levels of abstractions, in: *Embedded Software Design Proceedings of the second International Conference on Application of Concurrency to System Design* (June 2001).
29. G. Kahn, The semantics of a simple language for parallel programming, in: *Proc. of the IFIP Congress 74* North-Holland Pub, (1974).
30. G. Kahn and D.B. MacQueen, Coroutines and networks of parallel processes,in B. Gilchrist (ed.), Information Processing 77,North-Holland Publishing Co., (1977).
31. H.M. Taylor and S. Karlin, *An Introduction to Stochastic Modeling*, Third Edition Academic Press, (1998).
32. E.A. Lee and A. Sangiovanni-Vincentelli, A framework for comparing models of computation, IEEE Transactions on CAD 17(12) (1998).
33. J. Misra, Distributed discrete-event simulation, ACM Computing Surveys 18(1) (1986) 39–65.
34. A. Bonivento, L.P. Carloni and A. Sangiovanni-Vincentelli, Rialto: A bridge between description and implementation of control algorithms for wireless sensor networks, in: *Proc. of EMSOFT 2005* (Jersey City, NJ, USA, Sept. 2005).
35. A. Bonivento, C. Fischione, A. Sangiovanni-Vincentelli, F. Graziosi and F. Santucci, SERAN: A semi random protocol solution for clustered wireless sensor networks, To appear in: *Proc. of MASS 2005* Washington D.C., USA, (Nov. 2005).
36. Wei Ye, John Heidemann and Deborah Estrin, Medium access control with coordinated adaptive sleeping for wireless sensor networks, IEEE/ACM Transactions on Networking 12(3) (2004) 493–506.

37. J. Polastre, J. Hill and D. Culler, Versatile low power media access for wireless sensor networks, Sensys (2004).
38. T.S. Rappaport, *Wireless Communications* (Prentice Hall, Upper Saddle River NJ, 1996).

Alvise Bonivento is graduate student at the department of Electrical Engineering and Computer Science at the University of California at Berkeley. He holds a Laurea degree from the University of Padova in Telecommunication Engineering (2002) and a Master of Science in Engineering from the University of California at Berkeley (2004). His research interest is in communication protocols for embedded systems with emphasis on system level design and protocol synthesis for wireless embedded networks and wireless industrial networks. He is coauthor of several technical papers on major communication and embedded systems conferences.

Luca Carloni is an Assistant Professor of Computer Science at Columbia University in the City of New York. He holds a Laurea Degree Summa cum Laude in Electronics Engineering from the University of Bologna, Italy, a Master of Science in Engineering from the University of California at Berkeley, and a Ph.D. in Electrical Engineering and Computer Sciences from the University of California at Berkeley. At Berkeley Luca was the 2002 recipient of the Demetri Angelakos Memorial Achievement Award in recognition of altruistic attitude towards fellow graduate students. His research interests are in the fields of design technologies for electronic systems (with emphasis on component reusability, communication protocols, synchronization mechanisms, and low-energy architectures), design methodologies for fault-tolerant deployment of embedded software on heterogeneous and distributed platforms, computer architecture, integrated circuits, and combinatorial optimization. Luca coauthored over thirty refereed papers and holds one patent.

Alberto Sangiovanni Vincentelli holds the Buttner Chair of Electrical Engineering and Computer Sciences at the University of California at Berkeley. He was a co-founder of Cadence and Synopsys, the two leading companies in the area of Electronic Design Automation. He is the Chief Technology Adviser of Cadence. He is a member of the Board of Directors of Cadence, UPEK, a company he helped spinning off from ST Microelectronics, Sonics, Gradient and Accent, an ST Microelectronics-Cadence joint venture he helped founding. He is a member of the HP Strategic Technology Advisory Board and of the Science and Technology Advisory Board of General Motors. He consulted for many companies including Bell Labs, IBM, Intel, United Technology, COMAU, Magneti Marelli, Pirelli, BMW, Daimler-Chrysler, Fujitsu, Kawasaki Steel, Sony, and Hitachi. He is the founder and Scientific Director of PARADES, a European Group of Economic Interest supported by Cadence and ST Microelectronics. He is a member of the High- Level Group and of the Steering Committee of the EU Artemis Technology Platform. In 1981, he received the Distinguished Teaching Award of the University of California. He received the worldwide 1995 Graduate Teaching Award of the IEEE for "inspirational teaching of graduate students." In 2002, he was the recipient of the Aristotle Award of the Semiconductor Research Corporation. In 2001, he was given the prestigious Kaufman Award of the Electronic Design Automation Council for pioneering contributions to EDA. He is an author of over 700 papers and 15 books in the area of design tools and methodologies, large-scale systems, embedded controllers, hybrid systems and innovation. Dr. Sangiovanni-Vincentelli has been a Fellow of the IEEE since 1982 and a Member of the National Academy of Engineering since 1998.