Ph.D. Thesis

Fabrizio Milazzo

Fault detection and data prediction for WSNs

Abstract

In the last few years, Wireless Sensor Networks (WSNs) have been extensively used as a pervasive sensing module of Ambient Intelligence (AmI) systems in several application fields, thanks to their versatility and ability to monitor diverse environmental quantities. Although wireless sensor nodes are able to perform onboard computations and to share the sensed data, they are limited by the scarcity of energy resources which heavily influences the network lifetime; moreover, the design phase of a WSN requires testing the application scalability prior to actual deployment. In this regard, this dissertation focuses on *data prediction* to address such crucial tasks as prolonging the network lifetime and testing the WSN scalability. Nevertheless, the matter is particularly challenging as the real world measurements are influenced by unpredictable events that affect the sensor readings. To this aim, *fault detection* techniques help to identify corrupt measurements and to discard them before they are actually transmitted within the network, so they may be profitably used to improve the precision of the prediction models.

This dissertation describes the design of two software modules which address fault detection and data prediction and may be combined in a single software system for WSNs. The fault detection submodule classifies the sensed measurements as "corrupt" or "regular" by means of Bayesian Inference. The prediction submodule builds models for the monitored quantities and is also able to generalize them to unknown environments populated by virtual sensor nodes so it allows to test the scalability of the application for networks of different sizes. Prediction also allows sensor nodes to reduce their energy consumption as much as possible by fine tuning their sampling rate based on the accuracy of the predictors.

Experimental results show the capabilities of the proposed system to detect faults and to build reliable prediction models for some of the most common physical quantities for WSNs, namely light exposure, temperature and humidity.

Acknowledgments

I would like to thank my Advisor Prof. Giuseppe Lo Re and the colleagues Marco Ortolani and Alessandra De Paola for giving me the opportunity to complete my academic path with the Ph.D study. Their advices helped me a lot of times and made me able to understand that "writing" for the science is harder than it seems.

I would like to express my sincere gratitude to all my colleagues at the Department of Computer Engineering for their precious moral support and for all the funny moments I had with them.

Finally, I will always be grateful to my girflfriend and her parents, my parents, my brother, my uncle, my aunt and my cousins for their continuous encouragement in the worst moments and for giving me the force to end this research project.

Contents

List of Figures

List of Tables

Glossary

AmI	Ambient Intelligence
QoS	Quality of Service
WSN	Wireless Sensor Network
CPU	Central Processing Unit
SDP	Sensory Data Prediction
AFD	Adaptive Fault Detection
BS	Base Station
CH	Cluster Head
PDF	Probabilistic Distribution Function

Chapter 1

Introduction

The research field on Wireless Sensor Networks (WSNs) has been very active in the last few years and currently deals with a lot of issues related to the resource-constrained nature of their building blocks, i.e. the wireless sensor nodes.

The aim of this chapter is to introduce the reader to the concept of WSN and the fields it is commonly applied. Later, the most common issues the research has focused in this years will be presented as well as why they are important when designing a WSN. Finally, the chapter presents the logical architecture of the software system object of this dissertation, that focuses on some specific and challenging aspects of the WSNs and a high-level description of its inner components.

1.1 Motivations and Goals

The recent advances in electronics have triggered the use of cheap and small sensing devices, the so-called *sensor nodes*, able to sense many physical quantities as temperature, humidity, magnetic fields, pressure, light exposure, movements, mechanical stress levels and so on, and to perform small on-board computations.

Wireless Sensor Networks (WSNs) are made up of many sensor nodes, and are potentially able to provide computational capabilities comparable to the traditional wired networks. Thanks to their ability to sense the surrounding environment, perform distributed computations and wirelessly communicate the measurements to mass storage devices, they were massively used in many fields of the ICT like military, medical, ambient intelligence (AmI), security and so on; moreover, the presence of consolidated and standardized communication protocols as IEEE 802.15.4 [1], Wireless HART [2] and Zigbee [3] and a reprogrammable and lightweight operating system as TinyOS [4] made it possible to think to the WSNs as a standard *de facto* for all that ICT applications requiring a sensing infrastructure to monitor the external enviroment [5, 6].

Despite their advantages, WSNs present some issues that need to be careful addressed at design time before the actual deployment of the network. The works [7, 8, 9] have identified the following as the most important design factors that affect the performance of a WSN:

- Fault Tolerance: the network must be able to sustain its functionalities in presence of node failures and possibly to identify which failures have occurred.
- Energy consumption: sensor nodes are typically equipped with limited power supplies (1.5V batteries) that may not be replaced easily. Network lifetime may dramatically worsen if the issue is not carefully considered during the design phase.
- Computational constraints: the Central Processing Unit (CPU) of a sensor node has very limited computational resources. Distributed computation should be massively used to exploit as best as possible the computational capabilities of the WSN.
- Scalability: the performance of the application running over the WSN may drop as the size of the network increases and the issue should be clearly checked prior the deployment phase.
- Deployment costs: it is mainly affected by the nature of the environment over the network will be used.

In the last decade the approach used for assessing the performance of the WSN considered the development of a full functional prototype, and to actually deploy it into the environment. This is for instance the solution adopted for iDorm [10], a prototype for a student dormitory that allows the simulation of different everyday life activities. However, this approach is no more viable due to its inner deployment costs that may be prohibitive in complex environments, such as entire buildings; moreover, it does not allow to test application scalability, nor to evaluate the application behavior across different configurations.

An alternative approach consisted in simulating the whole control loop, from the physical to the application layer of the sensor nodes. The Intelligent Home [11], for instance, is a simulated testbed intended as a support for the development of multi-agent systems scattered over a WSN. The drawback of such an approach is that the whole application logic is in general not easily reproducible, and in particular it is difficult to capture the runtime overall behavior of the system. Nevertheless, early detection of design errors, and a fine tuning of critical factors, such as the position and number of sensor nodes in the various areas of the test site, may avoid subsequent, presumably expensive, re-deployment.

Currently several simulation frameworks exist for WSNs which provide controlled, and reproducible environments for tests, but they are not guaranteed to deliver fully reliable results, especially when the application logic is heavily sensitive to the actual sensor readings. In order to minimize the difficulties in porting simulated sensor networks to actually deployed systems, it may be advisable to use "real code" simulation tools, that run identical code in simulation and deployment, and this was the innovation proposed by TOSSIM [12], the TinyOS simulator.

The current state of the art solutions mix the benefits provided by a full-functional prototype to those of a network simulator. In particular, *hybrid simulators* have been profitably proposed as a way to generate reliable, and easily scalable applications by the interaction of virtual sensor nodes with real ones. The real nodes are used to anchor the network to the reality and to capture the dynamics of the environment they will be deployed; on the other hand, the use of simulated nodes allows to limit the deployment to just a minimal set of real nodes, which may serve as realistic data model generators to steer the behavior of their virtual counterpart and allow to evaluate some issues like fault tolerance, network lifetime, scalability and so on.

Although the current implementations of hybrid simulators are able to reliably represent many aspects of a sensor network, e.g. physics of transmissions, timed execution of the real code and so on, they are are not able to provide meaningful data for a given physical quantity and this fact limits the ability for the designers to evaluate the performance of the WSN. In order to implement an effective hybrid simulator, it may be advisable to develop models that allow to represent and predict data; as shown in the following chapters, such an ability allows to test the behavior of the WSN prior to its deployment as well as to virtually extend the coverage of the network after the deployment. Moreover predicting data may be used as a way to prolong the network lifetime in that sensor nodes will be required to sense the environment less frequently thus saving the energy for other tasks.

This intuition is motivated by the work of [7] that states that the main sources of energy consumption in wireless sensor nodes are the sensing, processing, and communication components; also the work [13] has shown that the transceiver represents the major drain, especially if compared to the CPU, which is the less energy-hungry component of the sensor node. The conclusion made by the authors is that energy efficiency may be achieved by exploiting the inherent spatial and temporal redundancy in data. In particular, many physical quantities as temperature, humidity, light exposure, pressure and so on, typically exhibit smooth variations, and change slowly over time [14] so predictive models may be used to generate realistic measurements from the environment and to reduce the need of sensing for the sensor nodes.

The problem of generating reliable prediction models is also related, to some extent, to the *fault tolerance* issues. Faults constitute a serious threat for the

application running over the WSN as they directly affect the Quality of Service (QoS) of the sensor network and cannot be neglected in the design phase. The faults occurring over a WSN influence different levels, namely the *Network*, *Node*, and *Sensor* one [15]. Network faults include loss of connectivity, routing loops, congestion and affect the timing of the messages exchaged among the node. Node faults are malfunctions of the main components of the sensor node, i.e. radio, CPU, battery and memory that lead to unexpected resettings, meaningless values of sensed data and poor quality of transmissions. Sensor faults only affect the quality of the data and are tipically caused by the ageing of the sensor or by the surrounding environment.

In the case of the traditional networks *remote monitoring* techniques may be a valid solution to discover faults and isolate malfunctioning nodes; however sensor nodes may be deployed in hostile environments, and may be potentially unreacheable by humans, making the approach not viable for the WSNs. Fault tolerance and fault detection represent a challenge for the WSN researcher and requires the adoption of sophisticated artificial intelligence techniques [16, 17, 18] that need to be added to the capabilities of the network. Building reliable prediction models only requires to focus on the sensor level faults which occur when gathered data does not provide a reliable representation of the monitored physical phenomenon; in this case, transmission and processing of these data clearly constitute a waste of energy and time and the overall ability of predicting data gets worse. On the contrary, an early detection of sensor faults reduces the amount of data processed by the higher-level applications other than providing high quality data to represent the related phenomenon.

This dissertation addresses some aspects of the fault tolerance (using fault detection), energy consumption and scalability (using prediction models) and proposes interesting solutions that help WSN designers to mitigate their related negative effects. The core of the dissertation consists into two software modules, namely *Adaptive Fault Detection* (AFD) and *Sensory Data Prediction* (SDP), that may be combined in a single software system FDDP (Fault Detection and Data Prediction) addressing fault detection and data prediction as shown in Figure 1.1.

The AFD module accepts as input the raw data sensed by the network nodes, exploits the inherent spatial and temporal relationships among readings to detect faults and produces as output the filtered data. Sensor nodes running the module may collaborate or not with their neighboring nodes to discover the erroneous data they sensed: if a node opts for cooperation, it is informed about the measurements in its proximity, thus widening the scope of the monitored phenomena; intuitively, such behavior should increase the chances of correct classification of the sensed measurements. On the contrary sensor nodes may choose to not send/receive readings from their neighborhood and relying only on the temporal relationships of their own sensed data to perform classification; for instance, such behavior

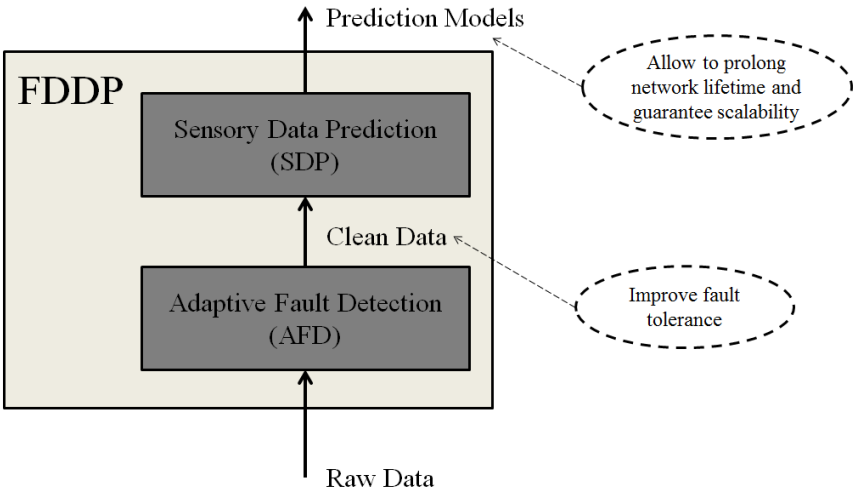


Figure 1.1: Components diagram of FDDP.

may occur when the nodes need to limit their computational resources, or when they discover that communications are not working properly (e.g. in the case of bandwidth saturation). Since sensor networks applications may require to be very reactive to the changes of the environment, also the module is made adaptive with respect to its inner response time and changes the topology of the network according to some user constraints. The detection problem is solved by making use of Bayesian Networks inference whose complexity is spread over the wireless sensor nodes. The distributed architecture makes the system highly reactive to changes in the monitored phenomena and allows to reduce the computational effort required to the single node. The Bayesian network structure is also adapted at runtime to reflect the choice of cooperating or not performed by sensor nodes; the more complex the structure, the higher the classification accuracy of the WSN; the simpler the structure, the lower the response time.

The SDP module accepts as input the filtered data and produces as output a set of prediction models that can be used both in the design or the running phase of the WSN for the purposes we mentioned above. The module exploits the temporal and spatial redundancy of the measured physical quantities and is designed to avoid as much as possible the need of communications and sensing for the sensor nodes, relaying the task of building prediction models to a central base station. Whenever the model appears to be reliable with respect to the sensed data, then the sampling rate of sensor nodes is lowered accordingly. The module also is able to generalize the prediction models built for a specific site and to port them to other target sites: this capability allows the WSN to increase its coverage without decreasing the precision of the prediction models and to avoid the need for the designer to actually deploy the real nodes over the target sites.

The validity of the system was assessed in many experiments which have shown that the prediction models effectively lower the number of required sensory readings as well as the amount of required transmissions without negatively affecting the reliability of the data; moreover, the effectiveness of the system to generalize the generated prediction models and to port them to other unknown sites was checked in a real scenario. Finally, the capabilities of the fault detection module were assessed in presence of a substantial amount of corrupt data and the results have shown its ability to identify faults by keeping as low as possible the required computational effort.

1.2 Contributions

The main contributions of the work presented in this dissertation are:

- The design of a novel method for detecting sensor faults that exploits the inherent temporal and spatial redundancy of the monitored physical phenomena. The algorithm works either in a centralized or distributed fashion and has an acceptable time and message complexity that makes it suitable for Wireless Sensor Networks.
- The implementation of the AFD module for detecting faulty data that is flexible to the needs of sensor nodes. Sensor nodes can choose how much time should be committed to the detection task; the more the time required by the algorithm the more is the accuracy of the detection. On the other hand, lowering the response time allows the network to be more reactive to the environment's changes. The algorithm is able to find the best trade-off between response time and detection accuracy also in the presence of external constraints imposed by the users.
- The design of mathematical models for representing and predicting the trend of the phenomena observed by a Wireless Sensor Network that allow to check the scalability of the network as well as to increase its coverage by adding virtual sensor nodes.
- The design and development of a novel scheme for finely tuning the sampling rate of sensor nodes based on the accuracy of the predicted data. This scheme allows to lower the energy consumption as the number of samples needed to reliably represent the monitored physical quantity is reduced if compared to a fixed sampling rate approach.
- The implementation of SDP, a software module that may be used jointly to the existing network simulators to check the scalability of the WSN and

to simulate networks where real and virtual nodes coexist, by the means of prediction models. It also offers support for reducing the energy consumption of the real sensor nodes using predicted data instead of the sensed one.

- The assessment of the results produced by the combination of AFD and SDP (i.e the FDDP system) for different scenarios where real and simulated data coexist.

1.3 Dissertation Outline

The remainder of the dissertation is arranged as follows: Chapter 2 describes the characteristics of the real data gathered by sensor nodes and reviews the current state of the art of fault detection methods and how data prediction may be used to lower energy consumption and to check the scalability of WSN applications.

Chapter 3 presents the implementation of AFD, the fault detection module which makes use of the inherent spatio-temporal correlations to recognize and filter out erroneous data.

Chapter 4 presents a general mathematical model for representing and predicting the physical quantities sensed by sensor nodes as well as a scheme that finely tunes the sampling rate of the nodes based on the goodness of the predicted data. Finally, the chapter proposes the implementation of the SDP module, able to simultaneously address scalability and minimization of the energy consumption.

Chapter 5 presents the experimental results achieved by using the software modules with real and simulated data. The aim of the chapter is to show that the prediction software module is able to port the prediction models generated for a reference site to target sites where few actual sensor nodes are deployed. The chapter also evaluates the performance of the fault detection module and how it allows to keep limited the prediction error in presence of a substantial amount of faults.

1.4 Publications

Parts of the work in this thesis have been published in several referred conference proceedings and journals:

- Alessandra De Paola, Giuseppe Lo Re, Fabrizio Milazzo and Marco Ortolani. Adaptable data models for scalable Ambient Intelligence scenarios. *International Conference On Information Networking 2011 (ICOIN 2011)*, Kuala-Lumpur, Malaysia, 2011

- Alessandra De Paola, Giuseppe Lo Re, Fabrizio Milazzo and Marco Ortolani. Predictive models for energy saving in Wireless Sensor Networks. *IEEE International Conference Symposium on a World of Wireless, Mobile and Multimedia Networks 2011 (WoWMoM 2011)*, Lucca, Italy, 2011.

- Giuseppe Lo Re, Fabrizio Milazzo and Marco Ortolani. Secure random number generation in wireless sensor networks. *4th International Conference on Security of information and Networks 2011 (SIN 2011)*, Sydney, Australia, 2011.

- Giuseppe Lo Re, Fabrizio Milazzo and Marco Ortolani. A distributed Bayesian approach to fault detection in sensor networks. *IEEE Global Communications Conference 2012 (GLOBECOM 2012)*, Anaheim, California, 2012.

- Alessandra De Paola, Giuseppe Lo Re, Fabrizio Milazzo and Marco Ortolani. QoS-Aware Fault Detection in Wireless Sensor Networks. *International Journal of Distributed Sensor Networks*, Volume 2013.

Chapter 2

Related Works

The purpose of this chapter is to review the current state of the art approaches dealing with the detection of faults and data prediction in wireless sensor networks.

First of all, it will be defined the concept of *data fault* and revised the main causes of corruption in WSNs as well as some common solutions to the issue. Later, the benefits of building prediction models in WSNs will be considered, and in particular why and how they may be used to help the WSN designer/users to address scalability and to reduce energy consumption. Finally, some implementations of prediction models in WSNs will be described highlighting their related pros and cons.

2.1 Addressing fault detection

The detection of *data faults* is a widely studied topic in WSN research but its correct definition slightly differs from that generally adopted in other fields. According to the classical point of view in statistics and data mining, a data fault corresponds to a data pattern not complying with a well defined normal behavior [19]. This definition however does not apply to WSNs, because in such context the network is unaware of the ground truth, thus the data fault definition can be modified as a "data pattern not conforming to the expected behavior of the monitored quantity".

The WSN literature extends such a definition, by providing a complete taxonomy of data faults occurring in wireless sensor nodes as well as their root causes. The main causes that force a sensor node to produce a corrupt measurement could be characterized according to the *system-centric* view of a fault as done in [20]:

- *calibration faults*: the ADC converter has a specific input-output curve that, due to sensor age, changes over time. As a result the sensed readings show a derivative of the phenomen different from the actual one;

- *connection/hardware faults*: example of such fault is a short circuit caused by a spill over the mainboard;
- *low battery*: this typically causes noisy and stuck-at (a constant value) readings.

The authors of the same work also propose a failure model for classifying data faults into three different categories that will be used as reference in the dissertation (Figure 2.1):

- *stuck-at*: a serie of values with zero variations (a);
- *outlier*: a measurement that significantly deviates from other sensor readings (b);
- *noise*: a series of data with a variance greater than expected (c).

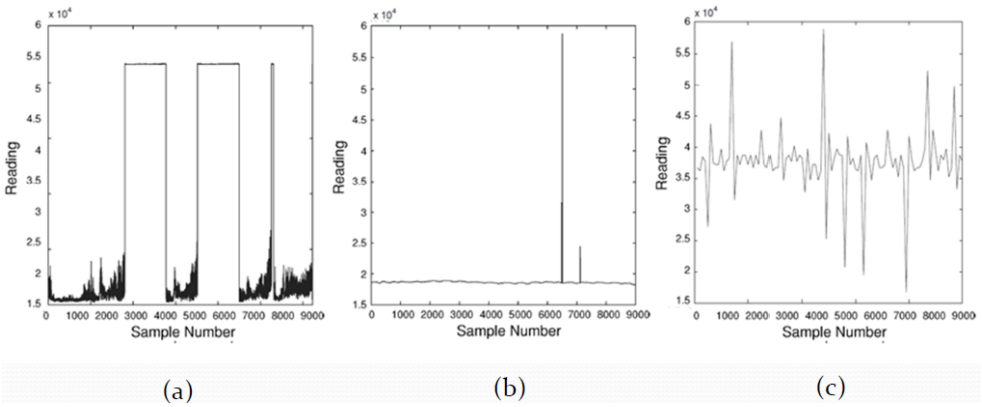


Figure 2.1: The main types of data faults in WSNs.

The current literature has developed many approaches to detect data faults classifying them with respect to the *architectural* and *methodological* viewpoint [15].

The architectural viewpoint distinguishes between *centralized* and *distributed* approaches. The work [21] considers a centralized multi-sensor fusion system that analyzes the impact of n sensors on the consistency of a non linear system of $n+1$ equations. The system is solved n times in a leave-one-out fashion by discarding the information gathered by one sensor at a time; the sensors that mainly affect the stability of the found solutions are chosen as the most discrepant ones and labeled as faulty. Also the work [22] implements a centralized technique that finds the subset ϕ of the network sensors maximizing the likelihood $p(D|\phi, \epsilon)$ of having a dataset of readings D given ϕ and a background knowledge ϵ. The main drawback

of such approaches is related to the NP-Hard computational complexity so the classification rates are limited by the resources at the base station.

The work [23] implements a distributed technique where each node is labeled as corrupt or not based on a consensus decision obtained by comparing its readings with the neighboring ones. If a node is recognized to be faulty it may discard its readings before transmitting them to the base station thus saving energy for the non-faulty readings; the work [24] is based on a similar process where the network is clusterized and the nodes of each cluster label the faulty nodes using consensus. The classification accuracy of such approaches is usually lower than the centralized ones; nevertheless, the fault detection occurs earlier with respect to centralized approaches, since it is not necessary to wait for the complete data transmission toward the base station.

As regards the methodological point of view, it is possible to distinguish between *Threshold based* and *Classification based* approaches.

The approaches of the former class use thresholds to perform fault detection. As an example, in the work [25] the network is initially clustered and for each cluster, based on the readings gathered, a certain number of *statistics* are computed; in a subsequent phase, the gateway node computes the so-called Inter-Cluster-Distance (ICD), the average $AVG(ICD)$ and the standard deviation $STD(ICD)$, for all of the underlying clusters. Finally, any cluster is labeled as faulty if its ICD is out of the range $AVG(ICD) \pm STD(ICD)$. Such approaches are generally not very expensive in terms of computational and time resources, but the classification accuracy is highly dependent on the chosen threshold values, thus the performance are quite unpredictable unless an extensive empirical tuning is performed in order to find the optimal threshold values.

The latter class avoids the use of thresholds, thus no human intervention is required to correctly tune the classifier. In particular *Bayesian Networks* and *Neural Networks* approaches belong to this category; in both cases, during an off-line learning phase, a model is automatically learnt from raw data, while an on-line algorithm exploits the learnt model in order to perform fault detection. The work [26] classifies the faults using Markov Chains (Bayesian Networks): in a first step the likelihood L of the last d readings $r(t)$ from time $t - d + 1$ to t is computed; then a new reading $r(t + 1)$ is classified as faulty if the new likelihood L' is less than a lower bound L computed during the learning phase. In the work [27] a Neural Network, based on Radial Basis Functions (RBF), is trained only using normal patterns of readings. When a new reading is supplied to the Neural Network, its output (ranging from 0 to 1) expresses the certainty that the reading is non-faulty; for a given set of N readings, the n istances with the minimum degree of certainty are labeled as faulty. The main drawback of such methods is their computational and time complexity but their classification accuracy is predictable and computable after the learning phase.

The software module implementing fault detection discussed in this dissertation, i.e. *Adaptive Fault Detection* (AFD), may be run either in a centralized or distributed fashion (the choice is left to the WSN designer) and uses a classification method based on Bayesian Networks. The centralized implementation does not pose attention to energy consumption or response time issues as the computation is done at the base station; its main benefit is a very high detection rate that is desirable in all those WSN applications not needing real-time computations. The distributed implementation has a lower detection rate but, at the same time, needs a lower response time and exploits cooperation among sensor nodes to spread the complexity of the classification problem over the whole network. As a consequence, in scenarios where the focus is on the response time, it is convenient to adopt such an approach to allow sensor nodes to be more reactive and to minimize the effort required by the detection algorithm, even if the classification accuracy may be lowered with respect to the centralized approach. On the contrary, in noisy scenarios, or when achieving a high classification accuracy is very important, sensor nodes relay on the computation done at the base station that will require a high response time but will be very precise in terms of detection accuracy.

An interesting feature of the distributed implementation of AFD is related to the possibility for the WSN designer to impose explicit constraints either over the response time or the classification accuracy; if no constraints are set, the algorithm will try simultaneously to maximize detection accuracy while minimizing the response time. The fault detection algorithm was formulated as a multi-objective optimization problem and, in particular the *Pareto Optimization* [28] technique was chosen in order to deal with different and typically contrasting objective functions.

Pareto optimization is versatile and currently used for a lot of applications in the field of WSN. The authors of [29] define a set of metrics for evaluating the network performance (at design time) with respect to *reliability*, *lifetime* and *coverage*; the experimental evaluation shows that the set of all possible solutions can be easily clustered and one of the identified cluster is Pareto optimal with respect to the considered qualitative dimensions. The authors of [30] propose the adoption of Pareto optimization to drive the clustering of network nodes minimizing the number of subcarriers allocated for each cluster and maximizing the system thoughput. The algorithm proposed in [31] has the goal of supporting WSN designer to find the optimal node placement; it is based on a multi-objective genetic algorithm, exploiting Pareto optimization, and it aims to find a trade-off between the transmission coverage and the number of deployed nodes. Finally, the authors of [32] cope with a multi-objective problem in an heterogeneous sensor network, and in that case the goal is to find the trade-off between the number of active sensor nodes that monitor a certain physical quantity and the overall network energy consumption.

2.2 Addressing Data Prediction

The correlation properties of the quantities monitored by a WSN make it possible to predict data at different time scales and with different degrees of precision. The analysis of the current state of the art in WSNs led us to conclude that data prediction can be effectively used to address many of the design issues cited in Chapter 1 and in particular to reduce energy consumption and to test/improve scalability of the applications running over the network. The following two sections will explain why data prediction is useful in the context of energy consumption and scalability; also a brief review on the practical implementations of data prediction in WSNs will be provided.

2.2.1 Data prediction for reducing energy consumption

The work [7] has identified the sensing, processing, and communication components as the main sources of energy consumption for a sensor node; the study [13] moreover has demonstrated that the most part of the energy is drained by the radio transceiver while the CPU is the less energy-hungry component and the results are shown in Table 2.1 and Table 2.2. The former table shows the drain of current for the devices running over a Mica2 sensor node: a great amount of current is drained by the radio component irrespective of its two working status (receiving, transmitting). The CPU also drains a comparable value of current when it is in the active status; nevertheless, for the rest of the time it works at four different sleep levels that gain very few current if compared to the radio component.

Table 2.2 provides an empirical proof that the energy consumption (milli-Joules) due to the radio is predominant for all that WSN applications needing communication. The table shows the data for seven NesC applications: the first and second application do not require communication so their energy consumption is due only to the CPU component. The remaining applications require communications and, as highlighted by the data, the energy consumption due to the radio is at least three times greater than that of the CPU.

Based on this information, in order to minimize the energy consumption, it may be advisable an approach to the design of WSN applications that trades communications for the computation. Many approaches in literature exploited this idea and as described in [33], they may be classified into three main categories: *duty cycling*, *mobility-based*, and *data-driven* based. Duty cycling approaches specifically target the optimization of the networking subsystem, mainly focusing on the implementation of efficient algorithms for controlling the sleep/wakeup schedule of the radio transceiver; such approaches are very effective for reducing the overall energy consumption but are not typically concerned with the data sensed by nodes. The second category is represented by mobility-based approaches, which allow for

Table 2.1: Drain of current for the components of a Mica2 sensor node.

Device	Component/State	Current
CPU	Active	7.6mA
	Idle	3.3mA
	Power Down	116μA
	Power Save	124μA
	Standby	237μA
Led	Each one	2.2mA
Sensor Board	Overall	0.7mA
Radio	Core	60μA
	Bias	1.38 mA
	Rx	9.6 mA
	Tx (-18dBm)	8.8 mA
	Tx (-13dBm)	9.8 mA
	Tx (-10dBm)	10.4 mA
	Tx (-6dBm)	11.3 mA
	Tx (-2dBm)	15.6 mA
	Tx (0dBm)	17 mA
	Tx (3dBm)	20.2 mA
	Tx (4dBm)	22.5 mA
	Tx (5dBm)	26.9 mA

Table 2.2: Energy consumption for seven different applications for WSN.

Application	**Consumption in mJ**					
	CPU		**Radio**		**Leds**	**Sensor board**
	active	idle	rx	tx		
Blink	0.37	601.6	0	0	196.2	-
CntToLeds	0.77	601.5	0	0	590.6	-
CntToLedsAndRfm	93	560.7	1651	130	589.6	-
CntToRfm	92.7	560.8	1651	130	0	-
RfmToLeds	82.9	565.2	1727	0.6	589.0	-
SenseToLeds	1.85	601	0	0	0	126
SenseToRfm	4.39	560.3	1651	130	0	126

the implementation of higher-level techniques, such as load balancing, data muling, or energy harvesting that however impose strict requirements on the needed hardware and are difficult to be implemented given the resource-constrained nature of the nodes. Finally, data-driven approaches are more tightly bound to the intrinsic nature of sensed data and often rely on the predictability of the physical quantities monitored, which are thus reliably represented through mathematical models. The main idea is to share a parametric mathematical model between sensor nodes and the base station from which drawing readings instead of sensing the external environment thus removing the need for the nodes to communicate their readings to the base station; whenever the shared models are no longer up-to-date, the model is recomputed by means of fresh readings and its parameters retransmitted back to the base station.

The SDP software module has been developed based on a data-driven approach but, differently from the other solutions proposed in the literature, it completely shifts the computation of the prediction models toward the base station. The sampling period of the sensor nodes is set based on the accuracy of the models while the base station draws samples from them instead of waiting for the fresh readings; the overall number of communications will be lowered resulting in an improvement for the network lifetime.

2.2.2 Data prediction for improving scalability

The main problem related to the design of a WSN is the unavailability of models for analyzing its behavior before the actual deployment. As discussed in Chapter 1 the issue may be addressed by adopting a totally "real" approach where, during a pre-deployment stage, the performance of a full functional prototype are evaluated to study the effectiveness of the WSN in the monitoring field: such solution is no more adopted as it involves high costs and does not allow to test scalability nor to test the behavior of the WSN for different environmental configurations. On the contrary a totally "virtual" approach would rise different issues, especially when the application behavior is heavily dependent on the actual sensor readings. In the last few years, the convergency toward solutions that fuse the benefits of the real (reliability of the results) and virtual (low costs for the deployment) approaches has produced the concept of the *hybrid simulation*.

According to [34] a hybrid simulator must be able, at least, to provide the following three features:

- Fidelity: The real world is represented through using mathematical models that simulate the layers of the WSN.
- Scalability: The simulator runs code emulation over the network nodes and it can be used to gather information about throughput, response time and

other QoS metrics in order to evaluate the scalability of the application for different network sizes.

- Heterogeneity: The simulator allows to mix sensor nodes with different hardware capabilities in terms of computational, energy and memory resources.

Currently, many network simulators matching the above requirements have been profitably used to ease the work of the WSN designer. The NS2 simulator [35] is a general purpose network simulator that implements several protocols (from the physical to the network layer) developed in C++. New features can be easily added given its modularity; in particular, models specifically designed for the WSNs may be included to improve the realism of simulations. The main drawback is the lack of support for native sensor code emulation, and in particular it lacks of the possibility to emulate NesC programs (the programming language of sensor nodes). OMNET++ [36] is a modular discrete event simulator implemented also in C++ that provides a GUI for virtual deployment of sensor nodes on the monitoring field. Finally, TOSSIM [12] is a simulator specifically developed for WSNs: its main advantage is the support for TinyOS code emulation and for different types of sensor node hardware; also a very precise radio model is provided. As a possible drawback, it is not able to provide an accurate model for simulating the network energy consumption.

Although the current state of the art simulators are able to produce accurate models for the radio channels as well as for the node's system, they do not provide mathematical models for realistically simulating the environmental phenomena surrounding the network. As a consequence, the ability of the designers for testing the scalability of the WSN application is very limited. In this regard, the SDP software module allows simulated sensor nodes to generate reliable readings based on the information provided by few actual nodes and can be used for two main purposes:

- Checking the application scalability *before* the deployment of the WSN by combining it with a hybrid simulator;
- Improving the application scalability *after* the deployment, by simulating larger networks with many virtual nodes that coexist with the real ones.

2.2.3 Existing solutions for data prediction in WSNs

The solutions proposed to solve data prediction in WSNs typically implement a *probabilistic* or a *statistical* approach [33].

Probabilistic approaches estimate a probability distribution function (PDF) based on the readings gathered for the observed phenomenon and then draw predictions using some predetermined sampling method. As an example, the work [37]

is based on Kalman Filters [38] with a user-specified accuracy, has a low computational cost but it is not suitable for non-linear physical phenomena. The authors of the work [39] propose to represent the missing or future sensory data $r(t)$ as a set of probabilistic distributions centered over an average value $M(t)$ and bounded by a pair of values $(L(t), U(t))$; the great advantage of this work is that can significantly reduce the communication burden in the sensor network as the base station draws samples from the PDF rather than waiting for the actual readings from the network.

Statistical approaches exploit the correlation properties of the physical measurements to forecast data. The authors of [40] propose a method that combines the information gathered by close nodes in order to build predictive models for reducing transmissions over the network. In their system, nearby nodes are grouped into clusters, and it is assumed that the cluster head acts as a representative for all nodes within its group. The intrinsic spatial correlation of data allows some of the nodes to go into a *sleep* state, while the cluster head keeps on sensing; the role of representative is routinely taken up by all nodes within the group. During its turn, each cluster head also computes a predictive model for data, which will be shared within its cluster, and used instead of actual sensing as long as it is deemed reliable. Such technique results in an overall reduction of the required transmissions. In [41], an autoregressive model is built using sensor readings. Initially, readings are collected until a buffer is filled; successively, each node computes a model for the sensed data and transmits only the model parameters back to the base station, thus effectively implementing a compression of data. The model is constantly checked for reliability against new readings; if a sufficiently large number of readings is recognized to fall behind a tolerance threshold within a given time window, the model is invalidated and recomputed by filling the buffer with fresh readings. However, if the readings are recognized as outliers, they are simply discarded, and the model will still be valid. The authors of [42] suggest to avoid the exchange of models through the network, by computing two separate predictors on the source and the sink node. Synchronization of such models is obtained by a minimal information exchange, consisting in the set of readings not satisfying a user-defined threshold, that signals the unreliability of the model computed at the source.

The SDP module implements a *statistical* approach as it is more suitable to exploit spatio-temporal relationships among the measurements. Sensor nodes are not required to build a local prediction model by delegating the whole computational burden to the base station (BS). They are just required to sense the environmental phenomena with a sampling rate depending on the reliability of the predictor and to communicate the readings to the BS. As a result the overall number of transmissions is reduced as well as the necessary computations; reliability of the computed model is checked whenever new data are received by the BS, which sets

the sampling rate of the source nodes accordingly (by decreasing it, if the model is reliable, and increasing it, otherwise). SDP also provides a practical solution to the problem of testing/improving the network scalability and this is obtained by means of prediction models that may be used either in the pre-deployment phase (testing) as well as after the deployment phase (improving).

Chapter 3

Adaptive Fault Detection

This chapter presents the software module *Adaptive Fault Detection* (AFD) developed to detect faults in the data gathered by the WSN nodes making use of the Bayesian Inference.

AFD implements the failure model depicted in Section 2.1 and shown in Figure 2.1. It exploits the idea that data is usually highly correlated in the space and time dimensions: stuck-at fault may be considered as a *temporal anomaly* and is usually detected by means of temporal series of data gathered by a single node; the noise or outlier faults instead mandatorily requires the comparison of a single sensory measurement with the data gathered by other nodes and they could be thought as *spatial anomalies*.

The module may be run either in a centralized or distributed fashion. In the case the centralized implementation is chosen (as an example clustered WSNs with the cluster head acting as the base station), sensor nodes are only required to send sensory readings to the base station that collects and labels them as corrupt or not. The only objective of the algorithm will be maximizing the correct classification rate.

If the algorithm is run using a distributed implementation, then the additional goal of minimizing the response time is taken into consideration as the overlay WSN application may pose strict real-time requirements. Sensor nodes are left free to choose if cooperate or not with their surrounding nodes. When the cooperation is chosen, the algorithm shows high classification rate and a high response time. Sensor nodes may also reduce the amount of cooperation, thus decreasing the response time but also decreasing the classification rate.

Chapter 5 will show that AFD achieves good classification performance if network nodes behave cooperatively (i.e. when they do exchange information on their measurements) and also achieves acceptable classification performance when sensor nodes behave non-cooperatively. The two constrasting goals of minimizing the response time and maximizing classification rate are fused through using the

so-called Pareto Optimization.

3.1 Mathematical model to discover data faults

This section describes the mathematical model and the algorithm developed for detecting faults in the data sensed by WSN nodes. For simplicity's sake the presentation will refer to a distributed implementation as the centralized one may be thought as a particular case where all the sensor nodes collaborate with each other.

Let us assume that a number of nodes deployed within the sensor field obtains readings about environmental quantities, such as temperature, humidity, or light exposure, and need to check if such readings are corrupt or not. Sensor nodes may try to classify their readings without communicating with each other by only using local knowledge (such as past readings, predictive models and so on); however they could also exchange sensed measurements in order to obtain additional valuable information about the physical quantity, thus increasing the probability that their own measurements are correctly classified.

The sensor field is assumed to cover a relatively narrow area so that the measurements sensed by different sensor nodes are likely to be correlated; in other terms, the variations due to the nature of the sensed physical quantity are relatively small with respect to nearby nodes.

The algorithm proposed here, for detecting corrupt sensory readings, exploits a Bayesian network distributed over sensor nodes. In such inference overlay network, the belief about the correctness of sensory readings flows thanks to communications among nodes. The use of a large Bayesian network allows to obtain a high classification accuracy, but, at the same time, it might cause a high response time; for such reason, the algorithm uses Bayesian networks with dynamic structure, in order to be able to adapt them according to the characteristics of the specific WSN deployment field. This adaptivity is obtained by allowing each sensor node to choose the set of neighbor nodes to cooperate with, and possibly also to choose to not cooperate at all. In the latter case, the sensor node tries to detect its own corrupt sensory readings by exploiting only temporal correlation among local measurements. On the contrary, if a node chooses to cooperate with its neighborhood, shared information is used as additional evidence, so that each node can also exploit spatial correlation to classify its own readings. These independent and dynamic decisions drive fault detection toward the configuration that represents the best trade-off among all possibly contrasting application goals, such as high classification accuracy and low response time. The distributed Adaptive Fault Detection algorithm (AFD) is composed by two main building blocks: a fault detection algorithm, based on Bayesian networks, and a QoS-aware optimization

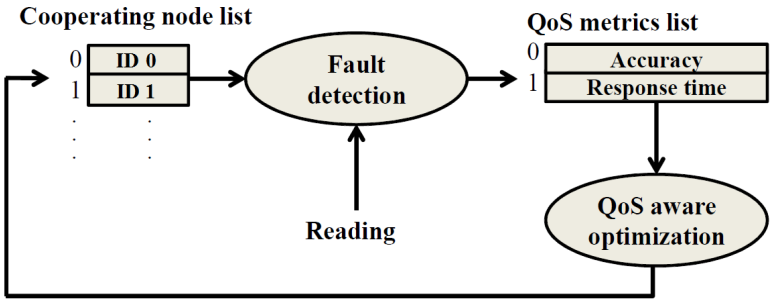


Figure 3.1: Block diagram of AFD algorithm.

algorithm, periodically affecting the structure of the Bayesian networks. It is worth noting that both the fault detection and the QoS-aware optimization are locally performed in every sensor node. Figure 3.1 shows the interaction between these two logical blocks, within a single sensor node. The fault detection block takes as input the sensory reading to be classified and the set of neighbor nodes to cooperate with. Such block, beside classifying the latest sensory reading, produces a set of QoS indices, namely the classification accuracy and the response time. Such indices are then used by the QoS-aware optimization block to modify the cooperating node list for the next step. Figure 3.2 shows the dynamic arrangement of the overlay communication graph in a simple running example. Let us suppose that at time $t = 0$ the overlay cooperation network includes all the sensor nodes in a single group, thus producing a very high classification accuracy. When the first optimization occurs ($t = T$), node 2 detects that there is some margin for reducing response time, thus it chooses to disconnect from node 1 and it consequently modifies its cooperating node list $CN(2) = \{4, 5\}$. After T more steps,

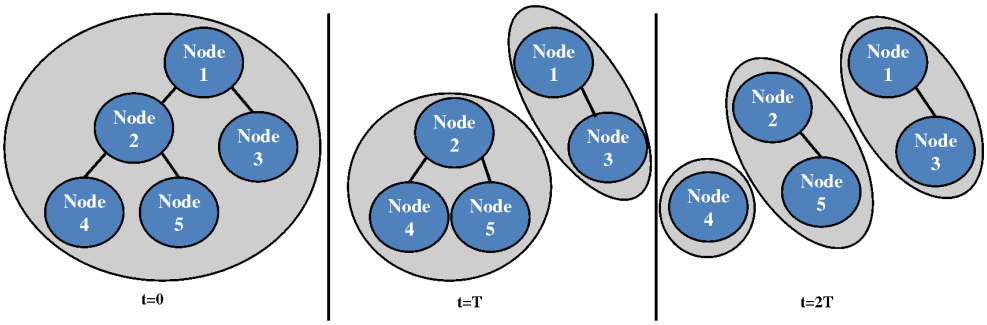


Figure 3.2: Example of the evolution of the communication graph at different time instants.

a new optimization occurs: because the classification accuracy is still sufficiently high, the node 2 chooses to disconnect also from node 4, thus its cooperating node list becomes $CN(2) = \{5\}$. The AFD algorithm performs analogously to an online clustering, since the update of the cooperating node lists modifies the overlay cooperation network. In order to ensure the convergence of the fault detection, it will be assumed for the rest of the paper that the underlying communication network is arranged as a tree so that every operation on the cooperating node list results in a set of tree arranged clusters.

3.1.1 Fault Detection

Fault detection is solved using distributed inference over the Bayesian network built on top of the cooperation network. The distributed inference algorithm takes as input a set of readings gathered by sensor nodes, and provides the classification of readings into corrupt or not. A first convergecast phase builds an initial estimate of the belief about reading classification, and a further broadcast phase refines such belief by adding information gathered over the rest of the network.

Bayesian networks are represented by a directed acyclic graph made up of random variables x_i, connected by directed links representing causal relations among variables. This model allows to take into account the probabilistic dependency between hidden random variables and observable random variables. The directed acyclic graph structure allows a simple computing of the likelihood function for the considered hidden variables. If $pa(x_i)$ denotes the set of parent variables for the random variable x_i, then the joint probability of the Bayesian network is:

$$p(x_1, ..., x_n) = \prod_i p(x_i|pa(x_i)). \tag{3.1}$$

In the implementation of AFD, each sensor node implements a small Bayesian network composed by one hidden variable c representing the (estimated) *class* of the current sensory reading and a set of observable variables representing the evidence probabilistically deriving from the true value of c. Observable variables can therefore considered as *features* related to the spatio-temporal correlations among readings.

Fault detection finds the combination of values for the hidden variable of each node, that maximizes its a posteriori probability, given the evidence. The Maximum a posteriori (MAP) approach is highly suitable to be implemented in WSNs, since it avoids the use of fixed thresholds and it is not too computationally expensive for sensor nodes.

Let $L = (l_1, ..., l_n)$ be the set of *local* observed variables, representing only the temporal correlation among the current and the past readings of a single node. If a node does not cooperate with its neighborhood, it can only exploit this evidence,

and the structure of its Bayesian network becomes a *Naive Bayes classifier* [43], as shown in Figure 3.3(a).

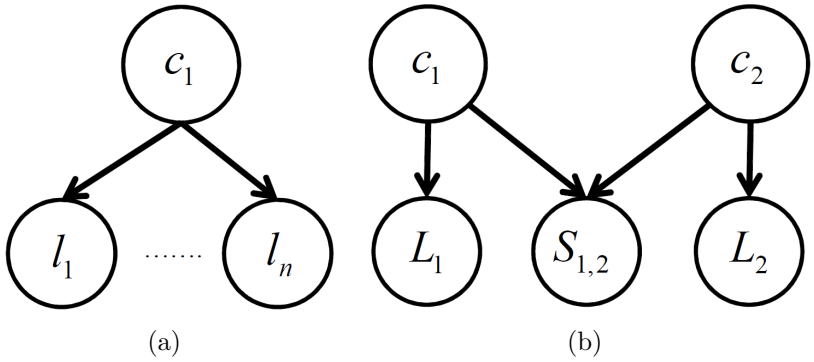


Figure 3.3: The Naive Bayes classifier, for a single node, including only local observed variables (a), and two Naive Bayes classifiers linked by shared observed variables (b).

Beside the local observed variables, the evidence variables include a set of *shared* observed variables, representing the spatial correlations among readings. If i and j are two cooperating nodes, then they share a set of variables $S_{i,j}$ that connect their two Naive Bayes classifiers by adding causal links from their hidden variables. The set of variables shared between two nodes i e j is a vectorial function of their last readings. Figure 3.3(b) shows two sensor nodes that linked their Naive Bayes classifiers through the use of shared variables; for the sake of simplicity, local and shared variables are grouped into the arrays of variables L and S.

There is a mapping between the current communication graph and the whole Bayesian network. The Bayesian network can be obtained from the communica-

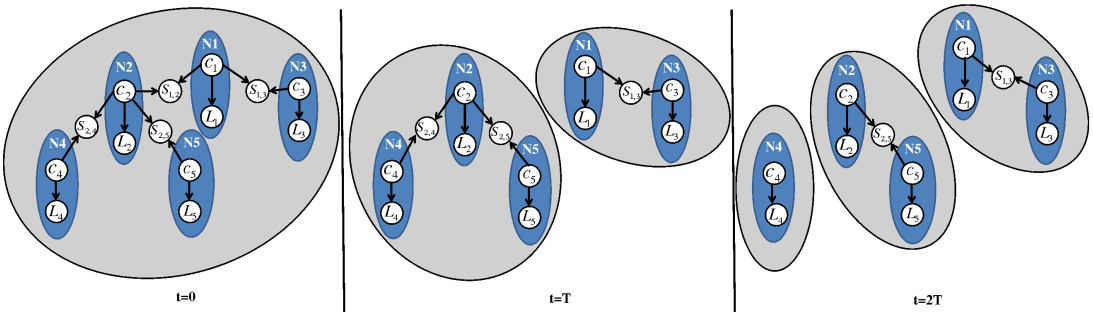


Figure 3.4: Evolution of the Bayesian network for a a simple network composed by five sensor nodes.

tion graph by replacing each sensor node with a Naive Bayes structure and each communication link (i, j) with a shared variable $S_{i,j}$ and by adding two causal links from hidden variables c_i and c_j to the shared observable variable $S_{i,j}$. Figure 3.4 shows how the whole Bayesian network evolves in the previously considered running example. Dark shadowed ovals represent Naive Bayes classifiers implemented by each node, while light shadowed ovals represent cooperating clusters. It is possible notice that every pair of nodes belonging to the same cluster, is connected through the shared variables set. The definition of the adopted fault detection method is completed by specifying the hidden and observed variables and how the MAP problem is solved.

Definition 1. *The hidden variable c takes values in the set* $C = \{$`outlier`, `noise`, `stuck-at`, `correct`$\}$.

According to the taxonomy proposed in [44]: a `outlier` fault is a reading whose value is out of range for the monitored physical quantity; a `noise` fault is a burst of readings whose variance is higher than the environmental one, and finally a `stuck-at` fault is a burst of readings that shows almost zero variations or, in other terms, that has a variance lower than the environmental one.

Definition 2. *Local observed variables are represented by a vectorial function of the last K readings of a single node and it is defined as* $L = \{l_1, l_2, l_3\} = \{$`inner-gradient`, `repetitions`, `variance`$\}$:

$$
\begin{aligned}
l_1 &= r_t - r_{t-1}, \\
l_2 &= \sum_{k=t}^{t-K+1} \mathbb{I}[|r_k - r_{k-1}| < \theta], \\
l_3 &= \sum_{k=t-K+1}^{t} \frac{(r_k)^2}{K} - \left(\sum_{k=t-K+1}^{t} \frac{r_k}{K} \right)^2 ,
\end{aligned}
\tag{3.2}
$$

where: r_t is the reading at time t; l_1 is the first derivative of r_t; l_2 is the number of consecutive readings, in a window of lenght K, that falls within a specific range θ and $\mathbb{I}[\cdot]$ is the indicator function (which is equal to 1 if the argument is true and 0 otherwise); l_3 is the variance of the last K readings.

Although the above set is not exhaustive and other features could be added to improve the classification accuracy, they was found sufficient to recognize the temporal correlations existing among readings.

Definition 3. *Shared observed variables are represented by* $S_{i,j} = \{s_{i,j}\} = = \{$`outer-gradient`$\}$:

$$s_{i,j} = r_t^i - r_t^j. \tag{3.3}$$

The shared variables set $S_{i,j}$ is, in principle, a vectorial function of the last K readings of sensor nodes i and j, but in practice it is a single function because it is sufficient to recognize spatial correlations; $s_{i,j}$ is simply the difference between the last reading sensed at node i and node j.

Definition 4. *The Maximium a Posteriori problem finds the solution array* $c^* = (c_1^*, ..., c_n^*)$ *that maximizes the joint probability of the Bayesian network*:

$$p(c_1, ..., c_n | L_1, .., L_n, S_{1,2}, ..., S_{n-1,n}) = \prod_{\substack{i \\ j \in CN(i)}} p(L_i|c_i) p(S_{i,j}|c_i, c_j) p(c_i), \tag{3.4}$$

where (i, j) are pairs of cooperating nodes, and $CN(i)$ is the cooperating node list of node i. The above equation is obtained by applying Equation 3.1 to the particular form of Bayesian network induced by AFD algorithm; it represents the joint probability of $(c_1, c_2, ..., c_n)$ of a single cluster of nodes and with slight variations could be extended for the whole WSN.

In order to solve this problem, the well-known "max-product" inference algorithm [45] was adapted to Bayesian networks. The code of the algorithm is provided at Section 3.3 and includes two main phases: (i) a convergecast phase, estimating the probability distribution over all possible classes for each sensed reading, and (ii) a broadcast phase, determining the final class labeling. Since the communication network is arranged as a tree, it is assumed that any sensor node is aware of being a leaf, a root or an intermediate node. In order to take into account the latency of the algorithm, it is also included a variable that keeps track of the number of hops (between any node and the root of the cluster) required for computing the class label.

The convergecast phase starts whenever fresh readings are to be classified. Each sensor node of the cluster computes the local features L_i as in Equation 3.2 and the *local belief* $\Lambda(c_i)$ as:

$$\Lambda(c_i) = p(c_i) p(L_i|c_i). \tag{3.5}$$

Such quantity represents the belief about the class of the sensory reading at node i, only considering local evidence. Successively, sensor nodes exchange their own readings with their direct neighbors, so they can compute also shared features as in Equation 3.3.

Each node i, with the exception of root node, sends a message to its parent j containing the *parent's reading belief* computed as follows:

$$\Phi(c_j)_{i \to j} = \max_{c_i} \phi(c_i, c_j). \tag{3.6}$$

The matrix $\phi(c_i, c_j)$ represents the joint belief about the classes of the readings of nodes i and j; such quantity takes into account *local belief*, *shared features*, and belief messages received from the i's children:

$$\phi(c_i, c_j) = p(S_{i,j}|c_i, c_j)\Lambda(c_i) \prod_{z \in CN(i)/j} \Phi(c_i)_{z \to i}, \tag{3.7}$$

where the productory is replaced by 1 if i has no children.

Root node r terminates the convergecast phase and computes its optimal class assignment as:

$$c_r^* = \underset{c_r}{argmax}\ \Lambda(c_r) \prod_{z \in CN(r)} \Phi(c_r)_{z \to r}. \tag{3.8}$$

Then, the root node starts the broadcast phase, where optimal classes c^* are propagated over the tree. Every non-root node i receives the label c_j^* from its parent j and computes its own optimal class as:

$$c_i^* = \underset{c_i}{argmax}\ \phi(c_i, c_j^*), \tag{3.9}$$

and then propagates such quantity to its children. This phase ends at leaf nodes and produces the solution of the MAP problem.

The performance of the AFD fault detection algorithm can be evaluated through the probability of correct classification p_{corr}^i, computed as:

$$p_{corr}^i = \phi(c_i^*, c_j^*). \tag{3.10}$$

3.1.2 Learning parameters

The learning procedure, needs to learn the conditional probability tables, namely $p(L_i|c_i)$ and $p(S_{i,j}|c_i, c_j)$, and the class priors $p(c_i)$, and consists in computing the joint probability for the whole Bayesian network.

Such Bayesian network model comprises two types of variables: hidden (c_i, with no parents), and observed (L_i, with one parent, and $S_{i,j}$, with two parents). By recalling the Equation 3.4 for the joint probability of the Bayesian network, it is possible to note that the parameters to be learned are the prior probabilities of the class label assignments and the conditional probabilities for local and shared features.

The computation of the conditionals is simplified by exploiting the independence properties of Bayesian networks as follows:

$$\begin{aligned} p(L_i|c_i) &= \prod_k p(L_i^k|c_i) \\ p(S_{i,j}|c_i, c_j) &= \prod_k p(S_{i,j}^k|c_i, c_j) \end{aligned} \tag{3.11}$$

The AFD module assumes the use of *supervised* learning with a training set made by a fixed amount of observed features with the respective actual label assignments. The computation of conditional probabilities $p(L_i^k|c_i)$, $p(S_{i,j}^k|c_i, c_j)$, and prior probabilities $p(c_i)$ is therefore carried out by using the frequentist approach.

3.1.3 QoS-aware Optimization

The fault detection algorithm is performed starting from a given structure of the communication network, defined in terms of clusters. The classification accuracy and the response time highly depend on the size of network clusters. In order to find the optimal cluster structure, sensor nodes are required to determine which neighbor nodes to cooperate with, on the basis of QoS indices associated with different configurations. The main goal is to find the network configuration representing the best trade-off among several application-driven QoS goals and constraints.

This QoS-aware optimization is performed through Pareto optimization, that allows to consider multiple objective functions, possibly contrasting and with non-comparable units of measurement.

The sensor network can be seen as a complex system where each agent interacts with the environment by taking decisions $d \in D$ at each time step of its lifetime. Before taking any decision, an agent can evaluate its goodness by means of a set of QoS metrics $m_d = (q_1, ..., q_n)$. In order to allow an agent to choose the better decision to be taken, it is required to define an order relation over the space of m_d; in particular, the Pareto dominance order relation was adopted. Let m_{d_1} and m_{d_2} be the quality metrics arrays respectively for the decisions d_1 and d_2, then m_{d_1} *Pareto dominates* m_{d_2} ($m_{d_1} \preceq m_{d_2}$) if each component k of m_{d_1} is lower than the corresponding component of m_{d_2}. Such definition, in a minimization problem, corresponds to the following equation:

$$m_{d_1} \preceq m_{d_2} \Leftrightarrow \{\forall k = 0, \ldots, n => m_{d_1}(k) \leqslant m_{d_2}(k)\}. \tag{3.12}$$

A Pareto optimal decision is defined as a decision that is not dominated by any other decision:

$$d^* = \{d_i \in D : \forall d_j \in D, d_j \neq d_i \Rightarrow m_{d_i} \preceq m_{d_j}\}. \tag{3.13}$$

The set of Pareto optimal solutions constitutes the Pareto optimal front.

In addition to multiple objective functions, a real application may be characterized by a set of constraints about QoS requirements. They can be taken into account by representing them as points in the QoS metric space, namely $v = (v_1, ..., v_n)$. A decision d is said to be *admissible* if and only if its quality metric array $m_d \preceq v$. This further check allows to eliminate possible Pareto optimal solutions that break at least one QoS constraint.

Each sensor node i has to periodically choose the subset of neighbor nodes to cooperate with. If $N(i)$ is its neighborhood set, then the output of the Pareto optimization is the decision d^* that drives the cooperating node list, $CN(i) \subseteq N(i)$, to the combination corresponding to the optimal value of the metrics array m_{d^*} that also satisfies the constraints array v.

The following of this section specify all the components of the QoS-aware optimization.

Definition 5. *The decision d for node i is defined as a pair of values $(action, id)$, where $action \in \{connect, disconnect, do\text{-}nothing\}$, and $id \in N(i)$.*

Each decision of the node i corresponds to a single atomic action affecting its cooperating node list, $CN(i)$. In particular i can choose to *connect* to or *disconnect* from one of its neighbors, or either to do nothing; this latter decision is taken when the current cooperating node list is Pareto optimal, and thus $CN(i)$ is left unchanged.

Definition 6. *The QoS metrics vector corresponding to a decision d is defined as $m_d = (q_{err}, q_{lat})$:*

$$
\begin{aligned}
q_{err}(t) &= \frac{\sum_{j \in CN(i)} \sum_{\tilde{t}=t-T+1}^{t} p_{err}^{j}(\tilde{t})}{|CN(i)| \cdot T}; \\
q_{lat}(t) &= \begin{cases} \max\limits_{j \in CN(i)} \{lat^{j}(t) + 2\} & \text{if } |CN(i)| > 0 \\ 0 & \text{otherwise.} \end{cases}
\end{aligned}
\tag{3.14}
$$

Such metrics are used for predicting the goodness of each possible decision d. The first metric, q_{err}, represents the average classification error of the nodes belonging to the chosen configuration of $CN(i)$ over a time window of length T; clearly, in order to compute such value, any node j of the cluster has to store the values of p_{err}^{j} for its last T readings.

In order to define the second metric, it is necessary to consider that the response time of the inference algorithm is proportional to the number of hops between i and the furthest node in its cluster; in the worst case i is a leaf of the tree representing the cluster topology, so the response time for node i will be at most equal to the maximum value of the response time ($lat^{j}(t)$) for nodes in $CN(i)$, plus the time needed by the two messages used for broadcast and convergecast on the link (i, j).

Definition 7. *The constraints array is defined as $v = (v_{err}, v_{lat})$.*

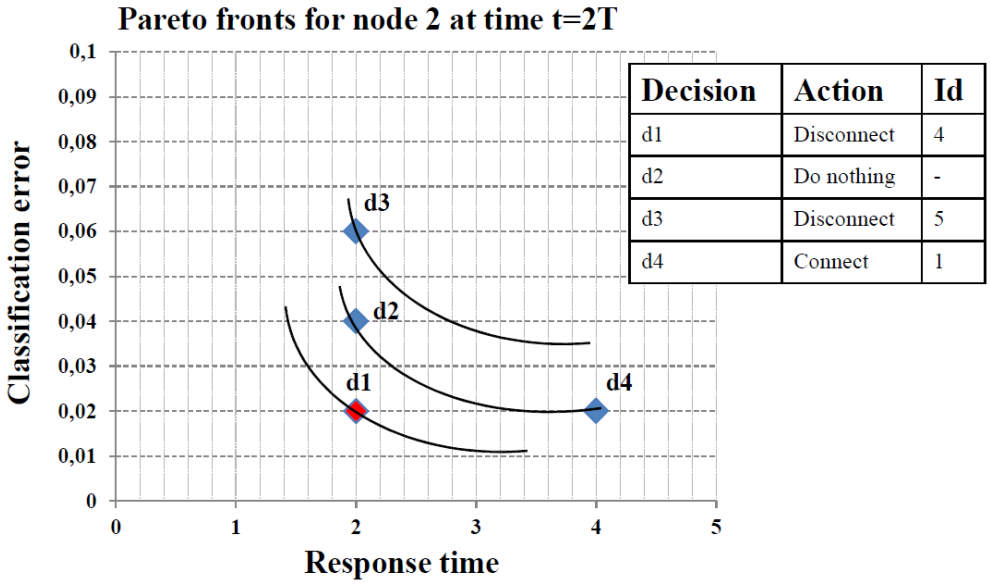


Figure 3.5: Example of a two dimensional Pareto optimization.

The satisfiability of a decision is verified by checking that the corresponding metrics vector is not dominated by the constraints, that is $(q_{err}, q_{lat}) \preceq (v_{err}, v_{lat})$. Such constraints are used as imposed upper bound for the values of the QoS metrics and they should be manually chosen by the application programmer.

In summary, node i evaluates the impact of changing its cooperating node list by forecasting the values of the future QoS metrics for the decisions of connecting or disconnetting from each node $j \in CN(i)$. Resulting solutions are filtered by cutting-off all those that are not admissible with respect to the specified constraints. The filtered solutions are ordered making use of definition 3.12 and the optimal front is found. If several solutions belong to the Pareto optimal front, a random decision among them is selected. Finally, it is also worth noting that it is possible that no admissible solutions are found after filtering; this occurs whenever the Pareto optimal front is placed beyond the constraints polyhedron. To cope with this situation, the constraints are relaxed by making all QoS metrics unbounded and then letting the QoS-aware optimization algorithm choose the Pareto optimal solution that is considered as the "less unsatisfatory" one.

Figure 3.5 shows, in the running example, the Pareto optimization performed by the sensor node 2 at time $2T$ for the classification error and the response time. Node 2 can choose among four different decisions, on the basis of its current cooperating list ($CN(2) = \{4, 5\}$); among such decisions, d_1 is the Pareto optimal one because its QoS metrics are lower than those of other decisions. It is also

worth noting that decisions are grouped into *Pareto fronts*; d_2 and d_4 belong to the same Pareto front, so they are Pareto equivalent.

In conclusion, AFD algorithm is composed by two main blocks: a fault detection block activated for each sensory reading, and a Pareto optimization modifying the communication graph over which the fault detection is performed, occurring only every T time steps.

Fault detection, described in section 3.1.1, takes as input the last sensory reading and produces as outputs the most probable class label, the corresponding probability of error and the number of communication hops. In order to achieve its goal, the fault detection block evaluates the set of local observable variables L_i; moreover, if $CN(i) \neq \phi$, node i exchanges its last reading within its cooperating neighborhood, and it receives their readings, so that all pairs of connected nodes are able to compute the shared observable variables $S_{i,j}$. Finally i runs the max-product to compute its optimal class label assignment and the probability of error p^i_{err}.

Outputs of the fault detection block are used to evaluate the current QoS metrics m, then sent to the Pareto optimization block, in order to compare the "do-nothing" decision with the other ones. Further inputs of the Pareto optimization are the current cooperating node list CN, used for evaluating the possible actions of connecting/disconnecting, and the array of constraints v used for cutting-off non-admissible decisions. Pareto optimization eventually produces as output the cooperating node list $CN(i)$ optimizing the QoS metrics; this list will be used for the next T time steps for performing fault detection.

3.2 Theoretical assessment

The aim of this section is to prove that the algorithm scales well with the size of the network. The mathematical notation contained in Table 3.1 will be used in the following.

Definition 8. *The fault-detection algorithm has memory complexity:*

$$O(k_f Z|C|^2) \tag{3.15}$$

Proof. The learning task stores a conditional probability table for every feature (local and shared) and a prior table for the class label. Let us suppose any node has at most k_l local features and at most k_s shared features; additionally let $k_f = max(k_l, k_s)$. Also suppose each feature is discretized into Z different values.

The required memory for such tables is bounded by: $k_f Z|C|$ for local features, $k_f Z|C|^2$ for shared features and $|C|$ for priors; by summing up the above terms, it can be easily seen that the overall required memory is $O(k_f Z|C|^2)$.

Definition 9. *The fault-detection algorithm has time complexity:*

$$O(|C|^2(2k_f + |N(i)|)) \tag{3.16}$$

Proof. Phase one involves a simple message exchange among neighbor nodes, and does not involve any computation.

Phase two is a convergecast-broadcast procedure. The convergecast message computation involves mono-dimensional maximizations (equation 3.6). To perform

Table 3.1: Legend of the notation adopted.

Name	Description	Name	Description
N	Number of network nodes	c_i	Hidden variable of node i in X
d	A decision	C	Set of possible values for the hidden variables c_i
Q_d	Quality array for decision d	D	Set of the possible decisions
q_{err}	Heuristic for predicting the classification error	q_{lat}	Heuristic for predicting the latency of the algorithm
$CN(i)$	Neighborhood set of node i in the cooperation network	lat	The latency experienced by the node in the last run of the algorithm
$N(i)$	Neighborhood set of node i in the communication network	v	Array of constraints
L_i	Set of local observed variables of node i	v_{lat}	Constraint on the latency of the algorithm
$l_{i,j}$	The j-th local observed variables of node i	v_{err}	Constraint on the classification error
k_l	Number of local variables $\|L_i\|$	k_s	Number of shared variables $\|S_{i,j}\|$
k_f	The maximum between k_l and k_s	M	Number of objective functions to be optimized
$S_{i,j}$	Set of shared observed variables of node i and j	T	Number of steps before a new optimization occurs
$s_{i,j,k}$	The k-th shared observed variables of node i and j	p_{err}	Probability of classification error

such operation the algorithm computes all the entries of the matrix $\phi(c_i, c_j)$ that are $|C|^2$.

The computation of a single value of $\phi(c_i, c_j)$ is a multiplication among $1 + 2k_f + (|N(i)| - 1)$ terms (see equation 3.7), that amounts to $2k_f + |N(i)|$ total multiplications. Time complexity for such round thus is $|C|^2(2k_f + |N(i)|)$.

During broadcast phase, the optimal class label assignment (see equation 3.9) is computed by a table lookup on $\phi(c_i, c_j)$, that requires $|C|$ operations. The time complexity for such round thus is simply $|C|$.

By summing all terms for broadcast and convergecast phases, the whole time complexity per sensor node results: $O(|C|^2(2k_f + |N(i)|))$, where all the irrelevant terms were discarded.

Definition 10. *The QoS-aware optimization algorithm has memory complexity:*

$$O(M|N(i)|) \tag{3.17}$$

Proof. The QoS-aware optimization algorithm stores the array of QoS metrics for all the admissible decisions. Each single node is able to choose among the following actions: (i) connect, or (ii) disconnect a node, (iii) do nothing; the total number of possible actions is equal to the size of the neighborhood in the communication network incremented by one. The worst case complexity is therefore given by $1 + |N(i)|$. Such a value is multiplied for M as it is the number of metrics corresponding to a single decision. The complexity is $O(M|N(i)|)$.

Definition 11. *The QoS-aware optimization algorithm has time complexity:*

$$O(M(|N(i)| + 1)^2) \tag{3.18}$$

Proof. The QoS-aware optimization algorithm orders the feasible solutions by means of the NSGA-II algorithm [46], whose complexity (in terms of number of comparisons) is $O(M|D|^2)$, where M is the number of objective functions (two in our case) and $|D|^2$ is the number of decisions to be analyzed. Since $D = |N(i) + 1|$ then the time complexity is $O(M(|N(i)| + 1)^2)$ per sensor node.

3.3 Implementation of AFD

The algorithm that performs fault detection may be run either in a distributed or centralized fashion. The centralized version is depicted in Algorithm 1 and does not take into account the response time, as the algorithm is supposed to be implemented at the base station. Moreover it assumes that the only goal is to maximize the classification accuracy so the cooperating node list of each sensor node is matched to the neighborhood set, i.e. $CN(i) = N(i)$; in other

terms, the Bayesian network is a connected and tree-arranged graph where the *Max-Product* algorithm is run to compute the best class labels assignments of the sensed readings.

Algorithm 1 AFD Algorithm (Centralized version)

Input Parameters:
I ▷ list of node identifiers
G ▷ connected and loop-free graph of the network nodes
R ▷ last K sensed readings for each network node
Output Parameters:
C^* ▷ array of optimal class labels
P_{err} ▷ array of error probabilities
Convergecast:
Compute local features L_i using Equation 3.2 for each i using R
Compute shared features $S_{i,j}$ using Equation 3.3 for each (i, j) using R
Run Max-Product$(I, G, L_1, L_2, ...S_{1,2}, ...) \rightarrow (C^*, P_{err})$
Output:
return C^*, P_{err}

The following set of algorithms implements the distributed version of the fault detection. Algorithm 2 may be seen as the main process of the whole distributed algorithm that accepts as input t, m, R, CN and v that are respectively the current time, the quality metrics for the current configuration of the cooperating node list, the last sensed K readings, the current cooperating node list and the array of the constraints for the objective functions. It provides as output c^*, p_{err}, lat and CN_{new} that represent respectively the most probable class for the last reading, its probability of error, the response time experienced and the updated cooperating node list (if changes occurred). Its code is splitted-up into two calls, i.e. one to the fault-detection algorithm and the other one to the Pareto optimization algorithm. It is worth noting that Pareto optimization only occur at regular time intervals of T steps, and this is to ensure stability of the cooperation or in other terms to avoid the network oscillates between several local minima too quickly.

The algorithm 3 implements the QoS-aware optimization and accepts as input M_d that is the array of QoS metrics for all the admissible decisions and provides as output d^* as the optimal decision. The algorithm generates the Pareto front by comparing the QoS metrics of the admissible solutions and retaining only those that are non-dominated by others. Then if many solutions belong to the Pareto optimal front, one at random is chosen and given as output of the algorithm.

The fault detection task, depicted in Algorithm 4, accepts as input i, CN and R that are respectively the node identifier, the current cooperating node list, and

Algorithm 2 AFD Algorithm (distributed version)

Input Parameters:

t ▷ the current time
m ▷ the current quality metrics
R ▷ the last K readings
CN ▷ the current cooperating node list
v ▷ the array of constraints

Output Parameters:

c^* ▷ the most probable class for the last reading
p_{err} ▷ the probability of error
lat ▷ the response time
CN_{new} ▷ the next cooperating node list

Fault detection:

Run *Fault detection (CN,R)* $\rightarrow c^*, p_{err}, lat$

Pareto optimization:

if t is multiple of T **then**
    for all $d \in D$ **do**
        Compute QoS metrics m_d using Equation 3.14
        Accept d only if $m_d \preceq v$
    end for
    if no admissible solutions found **then**
        Accept all d as admissible
    end if
    Run *QoS-aware Optimization*$(m_{d_1}, m_{d_2}, ..., m_{d_n}) \rightarrow d^*$
    Update CN according to the decision d^*
end if

Output:

return c^*, p_{err}, lat, CN

the last K sensed readings. It provides as output c^*, p_{err} and lat that represent respectively the class label assigned to the last sensed reading, the probability of error for the assigned class label and the latency experienced by the algorithm for computing the class label. The algorithm is divided into two main phases: convergecast and broadcast. The mathematical steps are described at Section 3.1.1.

Algorithm 3 QoS-aware Optimization Algorithm

Input Parameters:
M_d ▷ The array of QoS metrics for all admissible decisions
Output Parameters:
d^* ▷ The Pareto Optimal decision

Pareto Optimal Front Generation:
for all $i \in M_d$ **do**
    mark i as pareto optimal
    for all $j \in M_d, j \neq i$ **do**
        if $M_d(j) \prec M_d(i)$ **then**
            mark $M_d(i)$ as non-optimal
        end if
    end for
    if i is marked as optimal **then**
        add i to Pareto Optimal front
    end if
end for

Pareto Optimal decision:
if Many solutions belongs to Pareto Optimal front **then**
    set d^* to a random i of the Pareto Optimal front
else
    set d^* as the unique i of the Pareto Optimal front
end if

Output:
return d^*

Algorithm 4 Fault Detection Algorithm

Input Parameters:
i ▷ A node id
CN ▷ The cooperation network
R ▷ the last K sensed readings
Output Parameters:
c^* ▷ the optimal class label
p_{err} ▷ the probability of error for the chosen label
lat ▷ the latency (# of hops) experienced for computing c^*
Convergecast:
Compute local features using Equation 3.2 and local belief using Equation 3.5
for all $j \in CN$ **do**
    Send the last reading $last(R)$ to j
end for
for all $j \in CN$ **do**
    Receive the last reading of j
    Compute shared features using Equation 3.3
end for
if i is a leaf node **then**
    Compute the parent's reading belief $\Phi(c_j)$ using Equation 3.6
else
    Receive $\Phi(c_i)$ from children in CN
    Compute $\Phi(c_j)$ using Equation 3.6
end if
if i is not a root node **then**
    Send $\Phi(c_j)$ to parent j
else
    Compute optimal class label c^* using Equation 3.8
end if
Broadcast:
if i is the root node **then**
    for all $j \in CN$ **do**
        Send c^* as optimal class label
        Send 0 as latency lat
    end for
else
    Receive the optimal class label c^* and latency lat from parent node
    Compute optimal class label c^* using Equation 3.9
    Compute p_{corr} using Equation 3.10 and $p_{err} = 1 - p_{corr}$
end if
if i is not a leaf **then**
    for all $j \in CN$ **do**
        Send c^* as optimal class label and $lat + 2$ as latency
    end for
end if
Output:
return c^*, p_{err}, lat

Chapter 4

Sensory data prediction

The goal of this chapter is to present *Sensory Data Prediction* (SDP), a software module that simulates virtual portions of the WSN, whose behavior is strictly related to the actual trend of the physical quantities; the software module thus is able to manage a hybrid simulated WSN where a limited set of real nodes is augmented with a larger set of simulated ones. The development of such software module is aimed to ease testing of scale-sensitive WSN applications by allowing for virtual deployment of a large amount of sensor nodes with the freedom to choose the shape and size for the setting, providing a simple way to test the application behavior in qualitatively different environments. The adherence to reality is granted by the inclusion of real nodes, whose sensed data are used to generate predictive models for the actual physical quantities; flexibility is also taken into account by adapting the obtained models to simulate the behavior of virtual nodes.

A preliminary discussion will focus on the mathematical details of the models developed to predict data both in space and time and how to generalize them to unknown environments; later, it will be described the high-level implementation of the SDP module together with the related pseudocode diagrams.

4.1 Mathematical Model to predict data

The proposed mathematical models allow to simulate the behavior of environmental physical quantities with respect to both the spatial and temporal dimensions. First of all, this allows to generate sensory readings in places where no devices are actually deployed, enabling the simulation of environmental scenarios wider than those actually at researchers' disposal. Moreover, it is necessary for the adopted models to be tunable, so that they can be instantiated according to real past sensory readings, and adjusted with respect to on-line incoming ones. They also

need to be sufficiently generic so that they can be applied to a different monitored area than the one for which they have been built. As regards the energy saving aspects, the possibility of predicting data along the temporal dimension allows to reduce sensing and communications providing the basis for prolonging the network lifetime.

Although the overall hybrid simulation mechanism is generic and valid regardless of the specific physical quantity, the particular mathematical model, learning algorithm and mechanism to port a model from a monitored area to another, need to be selected, also taking into account the typical observed trends. More specifically, some criteria have to be met for our approach to produce reliable models:

- for each point in space, sensed data are temporally correlated so that it is possible to predict future values with sufficient precision;
- for each time instant, sensed data are spatially correlated, i.e. sensory readings in a given point can be estimated from the readings of close sensors;
- for each monitoring area, the temporal and spatial trend of the considered physical quantity can be derived from those of a similar, close monitored areas.

The scenarios described in this dissertation mainly consider physical quantities as temperature, relative humidity, and light exposure, as they represents typical environmental features considered by WSN applications. In most indoor and outdoor environments, those quantities typically present a periodic pattern, with a similar trend for all sensor nodes deployed throughout the monitored area; intuitively, this is a good hint that it may be feasible to learn mathematical models able to describe and predict them, and such models may also be reused and applied to other environments with similar physical characteristics as the one considered as reference. Clearly, each physical quantity requires individual modeling, i.e. different models will be independently derived for temperature, humidity, and light respectively; in the following, the function representing the spatial-temporal trend of a certain physical quantity will be indicated by $F(x, y, t)$.

The model must be formulated so as to capture information about the physical location of the sensors as well as potential temporal correlations over several measurements. This can be highlighted by explicitly considering a different function $f_{x_i,y_i}(t)$ per each node i, which represents the predictive model for the measurements of node i with validity of 24h.

The goal is to extract a global model providing measurements for each point in a reference site in order to "feed" the virtual sensor nodes with realistic values; the function $F(x, y, t)$ must be inferred in order to include all the information provided by the $f_{x_i,y_i}(t)$ functions. Also $F(x, y, t)$ must be able to generalize the behavior of the phenomenon to points where no sensor nodes are actually placed.

Basically, the idea is to compute $F(x, y, t)$ through using a simple interpolation of the values predicted by the local $f_{x_i,y_i}(t)$ functions. The method chosen for interpolation provides the following properties:

1. $F(x, y, t)$ is smooth, i.e. it is continuous and belongs to the class C^∞;
2. the interpolated values fall within the same range as the values predicted at each point in time.

The first property reflects the fact that the monitored physical quantities do not naturally present discontinuities in the environments. Even though light exposure may occasionally contradict this assumption in localized points (spots), this is usually not particularly relevant to WSN applications; moreover, strict modeling of this phenomenon would prevent from formulating a general approach valid for all the considered physical quantities at the same time.

The second property aims to ensure that $F(x, y, t)$ does not present outlier values devoid of any physical interpretation; in particular, the property ensures that no unrealistic values are generated in points where no sensors are placed. Highly noisy readings from one sensor, however, might heavily influence the global shape of $F(x, y, t)$, so it will be assumed without loss of generality, that measurements are not affected by environmental noise. The problem of filtering out noisy readings before prediction is object of the AFD module, described in Chapter 3.

Because of these considerations, it was used a normalized linear combination of the $f_{x_i,y_i}(t)$ functions as a spatial interpolator, as shown by the following equation:

$$F(x, y, t) = \frac{\sum_{i=1}^{N} w_i f_{x_i,y_i}(t)}{\sum_{i=1}^{N} w_i}, \tag{4.1}$$

where the summation is computed over a set of N deployed sensors, $w_i = e^{-d_i}$, and $d_i = \sqrt{(x - x_i)^2 + (y - y_i)^2}$, so that each value given by this model is computed by taking into account all the deployed sensors, but with higher importance given to the ones closer to the considered point.

The next step consists in determining how to adapt the environmental models built for one known site (*reference site*) to different areas presenting similar characteristics (*target sites*), as may be the case for different rooms in the same office building. As stated in the previous chapters, such an ability would make it possible to test the scalability of the WSN application for network of different sizes; the WSN designer only needs to deploy some real sensor nodes over the testing environment and then adding the virtual nodes to extend the size of such *hybrid* network.

A naïve solution may consists in superimposing the model computed for the reference site to other different and target sites; anyway this approach does not work, because of the differences in shape, size, light exposure and so on between the reference and the target areas. Given two different monitoring sites, one taken as reference and the other as target site, the knowledge of the pattern of the physical quantities in the reference site however supplies meaningful hints on the correct trend of the same quantities in the target site, provided that the models are properly transferred.

The only assumption needed to port a model from the reference to the target site, is that the considered physical phenomena present some underlying similarity that is preserved across the space and the time; in particular, the same pattern for temperature, humidity or light may be observed in different areas, despite the differences in the specific measured values.

The problem is formulated as the search of the set of geometrical parameters, that characterize the corresponding transformation, defined as follows:

- *intrinsic* parameters, which control the shape of the model functions $f_{x_i,y_i}(t)$ for the reference site and allows to predict data along the temporal dimension;
- *extrinsic* parameters, which control how the model computed for the reference site may be mapped onto a different target site, and basically summarize all the required geometrical transformations (translations, rotation, stretching, and so on) on the models. They allow to predict data along the spatial dimension.

4.1.1 The Intrinsic Parameters

The intrinsic parameters model the shape of the environmental functions representing the observed physical quantities within the area of the reference site.

A generic function, modeling the readings of sensor i may be approximated by a series of M Gaussian functions depending on the previous sensors readings:

$$f_{x_i,y_i}(t) = \sum_{j=1}^{M-1} w_j \mathcal{N}(t|\mu_j, \sigma_j^2), \tag{4.2}$$

where the spatial coordinates of sensor i are not explicitly indicated. The intrinsic parameters of the model are computed by means of square error minimization [47] and use the readings collected in the previous time interval as training set.

The approach also takes into account the natural periodicity of environmental phenomena, so the model for a given sensor learnt for a time interval Δt_{old} will be used also as predictor for time interval Δt_{new} after proper scaling and translation:

$$f^{new}_{x_i,y_i}(\alpha, \theta, t) = \alpha \left[f^{old}_{x_i,y_i}(t) - \theta \right] + \theta. \tag{4.3}$$

Parameters α and θ can be computed for a new time interval Δt_{new} by considering a new observation set S^{new}, and a model $f^{old}(t)$ for a past time interval Δt_{old}, through least squares optimization:

$$\underset{\alpha,\theta}{\mathbf{argmin}} \sum_{t \in \Delta t^{new}} [f^{new}_{x_i,y_i}(\alpha, \theta, t) - S^{new}(t)]^2. \tag{4.4}$$

Having computed the intrinsic parameters of the models of the sensor, it is possible to implement a predictor for each of them; such predictor should not modify the overall shape of the underlying environmental function, although at the same time it should adapt to transient climatic changes (e.g. a rainy day when the average light exposure is lower than usual, or a sudden drop in temperature).

The presence of such predictors makes it possible to lower the network's energy consumption as the real measurements are replaced by simulated ones and sensor nodes are not required to communicate them toward the base station. The base station adaptively controls the sampling rate of any sensor node by decreasing it when the absolute difference between predicted and actual measurement is lower than a given threshold, and increasing it otherwise. Such behavior makes the sampling process faster when the estimated model is not sufficiently reliable; intuitively, the fact that the model becomes less reliable may be due to the presence of sudden bursts of high-frequency data (noise, or sudden variations) which should trigger a greater accuracy for the model itself. On the other hand, sensor nodes are allowed to save energy (relative to sensing, and network transmissions) because the sampling period is reduced when the model fits well with the actual data.

4.1.2 The Extrinsic Parameters

Models computed for the reference site must be extended to be ported to environments with slightly different characteristics. For each of the environmental functions relative to the considered phenomenon, we assume that the value in at least one specific spatial point $\mathbf{p} = (x, y, t)$ is known; the aim is to compute the transformation that maps the coordinate space of the reference site into a new coordinate space for the target site; each point $\mathbf{p} = (x, y, t)'$ would then be mapped onto $\mathbf{P} = (X, Y, T)'$. Such mapping may be formally defined as $\mathbf{P} = \mathbf{M_S}\mathbf{M}_\phi\mathbf{p}$, where $\mathbf{M_S}$ depends on the scaling parameters s_x and s_y relative to the spatial dimensions of the reference and target sites, while $\mathbf{M}_\phi$ depends on the parameter ϕ controlling the spatial rotation (useful, for instance when considering different light exposure between the reference and the target sites).

The overall transformation matrix may thus be written as:

$$\mathbf{M} = \mathbf{M_S}\mathbf{M}_\phi, \tag{4.5}$$

and the new environmental function for the target site will thus be given by:

$$F^{tar}(\mathbf{P}) = \beta(t) \cdot F^{ref}(\mathbf{p}) = \beta(t) \cdot F^{ref}(\mathbf{M^{-1}P}), \tag{4.6}$$

where the $\beta(t)$ weight (generally time-dependent) stretches the transformed function to better fit the target environment, and needs to be estimated as will be explained in the following.

In order to estimate optimal settings for the extrinsic parameters it is possible to use a simple, empirical approach, or an automated method based on the measurement of a small set of sensory data from the target environment. For the rotation and scaling parameters the former method was preferred, as support from the human operator seemed reasonable; the required effort is minimal and basically consists only in computing the size and orientation of the target environment, in order to produce the mapping function.

The parameter $\beta(t)$ is computed by exploiting a few sensory readings obtained by placing a minimal set of sensor nodes to be used as *probes* in the target environment. The weight $\beta(t)$ is a proportionality factor that fits measurements predicted for the reference environment (Equation 4.1) into the probe measurements sensed in corresponding points of the target environment, and basically acts as a time-dependent stretching factor.

The $\beta(t)$ stretching factor is estimated incrementally. Starting from the measured value $S_{X_i,Y_i}(t)$ for sensor i in location (X_i, Y_i) at time t in the target environment, and, knowing the mapping matrix $\mathbf{M}$, the system infers the prediction for the corresponding point $s_{x_i,y_i}(t)$ in the reference site. Assuming that the actual measurement in the target site differs from the one estimated through this mapping by a stretching factor $\beta(t)$, the estimation of $\beta(t)$ at various time instants is computed by minimizing the following error function with respect to β:

$$E(\beta) = \sum_{\mathbf{P} \in T_r} \{S_{X_i,Y_i}(t) - \beta s_{x_i,y_i}(t)\}^2 \tag{4.7}$$

where the summation runs over all the probe points $\mathbf{P} = (X_i, Y_i, t)'$ in the target region T_r.

4.2 Implementation of SDP

The implementation of SDP assumes that the sensor network is organized according to a cluster-based topology, as depicted in Figure 4.1, where leaf nodes forward sensed data towards their representative cluster head, which in turn computes a model for the considered physical quantity, and sets the sampling rate for each of its children nodes in order to keep the model reliable over time and to minimize the energy consumption due to redundant transmissions.

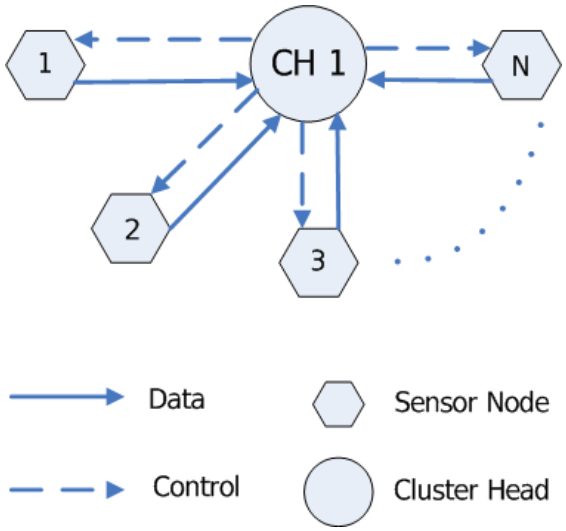


Figure 4.1: Example of a star topology cluster, highlighting the nature of the communications between the cluster head and each of the nodes running SDP.

The role of cluster head is assigned to a powerful wireless sensor node, the so-called *micro-server* (e.g. a Stargate node), which is typically equipped with larger amounts of memory, and offers more computational power. Given the characteristics of the considered quantities, it is expected that some kind of periodicity is loosely present in data, so the underlying assumption is that the overall "shape" of the trend representing each physical quantity is preserved over time.

The software module has been realized to extend the capabilities provided by the current state-of-the-art WSN simulators that lacks of the capability to represent the environmental phenomena through mathematical models. For the purposes of the dissertation SDP was integrated and tested on top of the TOSSIM WSN simulator [12]. The choice of TOSSIM as the underlying framework brings several practical advantages. For instance, no modification is required to existing NesC code, as SDP acts as a transparent software with respect to other applications; more importantly the very same code may be executed both on virtual and on real sensor nodes, which helps during the design and pre-deployment phase to ensure scalability of the WSN.

Figure 4.2 presents a high-level structure of the resulting hybrid simulated WSN [48]. Some of the virtual nodes are logically coupled to real ones; such hybrid (*shadow*) nodes represent the projection of real ones into the simulation and may be regarded as wrappers whose main purpose is to act as interfaces toward their real counterparts, while appearing identical to other virtual nodes from the simulator point of view. The main function of shadow nodes is to collect sensed data from the real world, and to re-route communication from virtual nodes to actual ones.

The interaction between shadow and real nodes is physically implemented at the base station where the simulation framework is running.

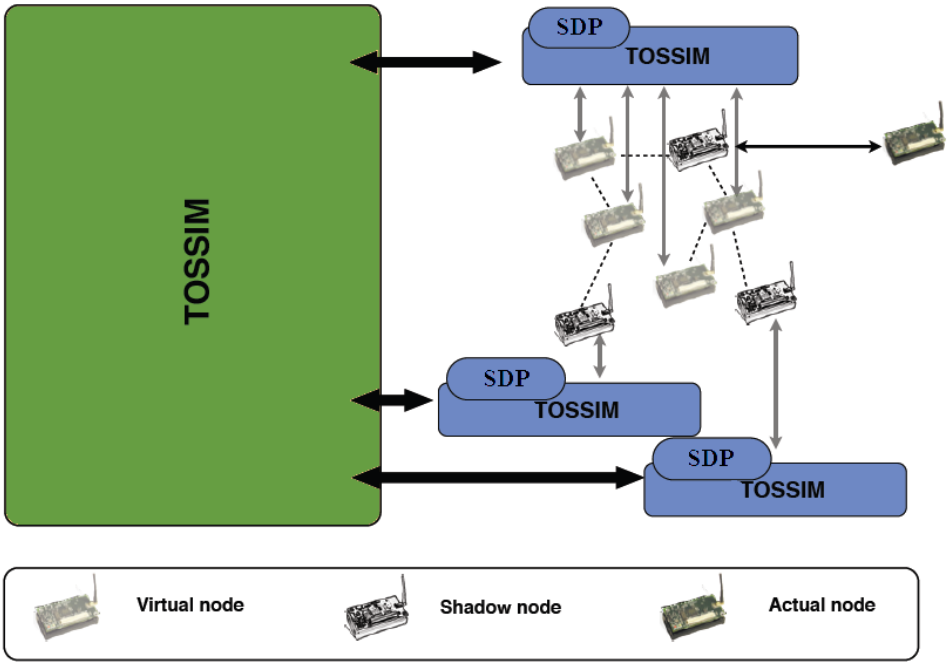


Figure 4.2: Example of a hybrid simulated WSN.

The use of TOSSIM solves some issues related to the coexistence of virtual and real nodes in a hybrid simulation: for instance, it is important that the simulation execution time reliably mirrors real execution time, especially in order to preserve causality; some kind of coordination must also be ensured between virtual and real nodes. To the best of our knowledge, TOSSIM provides acceptable performances in terms of soft real time constraints, and probabilistic end-to-end delay guarantees thus allowing SDP to be implemented independently from such low levels issues. In the context of the WSN applications, the soft real time constraint is totally acceptable with respect to simulation reliability. Sensing rates in the order of milliseconds or greater are generally totally acceptable for the kind of physical quantities involved in WSNs, and they cannot interfere with the realism of a simulation run.

The energy conservation feature, according to the taxonomy proposed by [33] and represented in Figure 4.3 is implemented by following a *data-driven* approach exploiting *adaptive sampling* and *data prediction* through a *statistical approach*.

The skeleton of SDP may be seen as the combination of three different algorithms that in sequence: (i) estimate the intrinsic parameters and recompute the sampling period of the sensor nodes, (ii) compute the environmental function

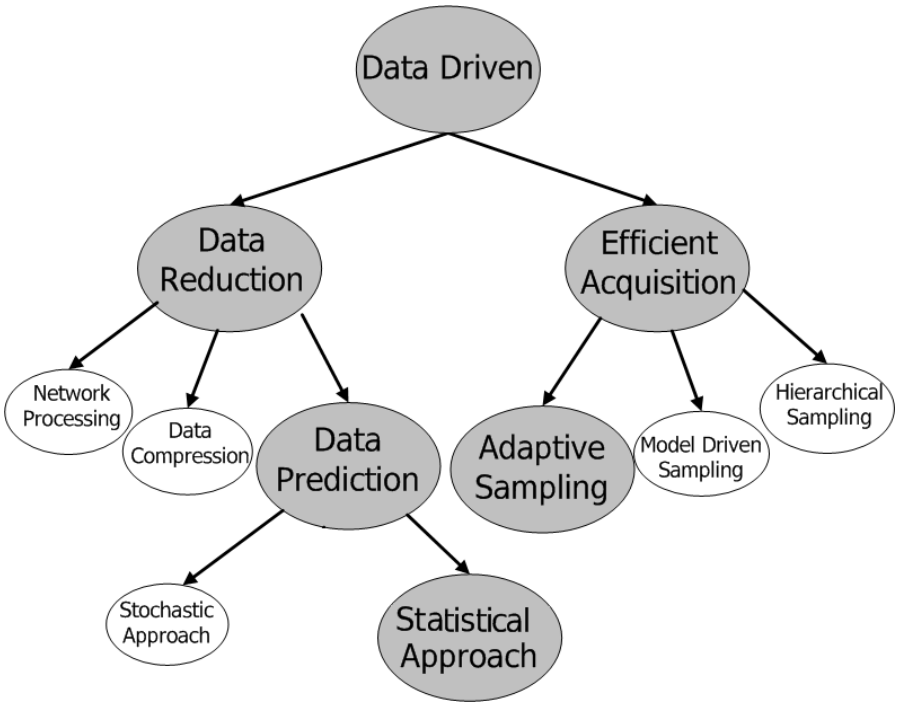


Figure 4.3: Categorization of data-driven approaches. Shadowed ellipses highlight the methods exploited in SDP.

$F^{ref}(x, y, t)$ and finally (iii) generalize the computed model by means of the extrinsic parameters. For convenience of the reader all the parameters used in the algorithms were arranged in Table 4.1.

Table 4.1: Intrinsic and extrinsic parameters.

	Parameter	Description
Intrinsic	w_j, μ_j, σ_j^2	Parameters of j^{th} Gaussian of the mixture.
	α, θ	Reshape parameters: allow to deform the previously fitted model in a new fitted model given the new measurements
Extrinsic	s_x, s_y	Scaling parameters: allow to resize the reference site into target site dimensions
	ϕ	Rotation parameter: it takes into account different light exposition between reference and target site
	$\beta(t)$	Scaling parameter: allows to deform the reference site fitted model in the target site fitted model

4.2.1 Estimating the intrinsic parameters

The process needed for computing the intrinsic parameters and, based on them, the resulting environmental function from which drawing predictions is summarized in Figure 4.4. It shows a schematic description for the construction of the interpolated environmental functions; models learnt from past measurements (*Fitted Models*) and current measurements (*Sensor Measures*) are used to construct a predictor of the various environmental functions. The *Spatial Interpolator* merges the different predictor functions through Equation 4.1 and builds the environmental model $F(x, y, t)$ for each of the phenomena monitored.

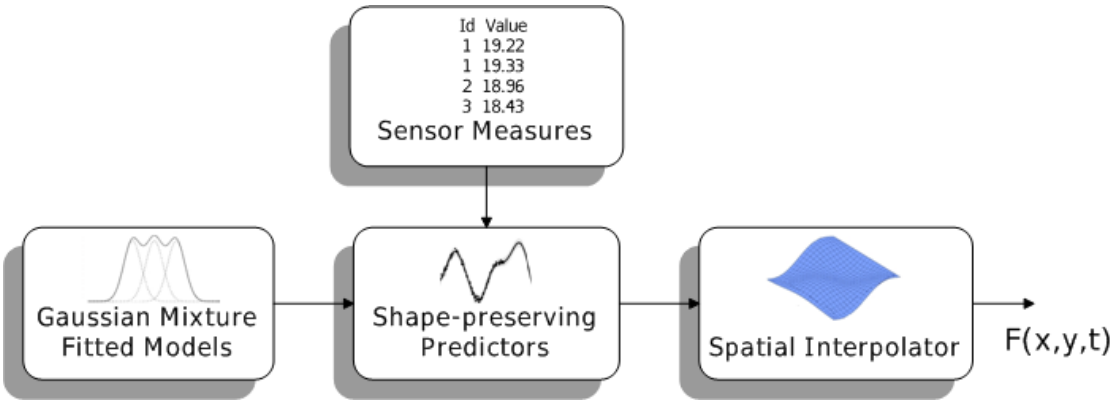


Figure 4.4: Generation of the environmental function for the reference environment.

The entire model $F(x, y, t)$ is therefore computed and updated on the cluster head through an iterative algorithm that accepts as input a fitting model $f^{old}(t)$ for the observed set in a past time interval Δt^{old}, and computes the new model $f^{new}(t)$ in an on-line fashion. The raw samples gathered from the sensor nodes are iteratively processed at the cluster head to refine the estimated values of α and θ.

The iterative algorithm for estimating the intrinsic parameters starts by initializing all the necessary variables:

$$\theta \leftarrow 0, \quad \alpha \leftarrow 1, \quad T \leftarrow 1, \quad R \leftarrow \emptyset,$$

where T is the factor controlling the sampling rate, and R is the set of the current sensory readings. The pseudocode for iteratively computing the intrinsic parameters for the current time interval is shown in Algorithm 5, where the threshold τ represents the maximum tolerable prediction error, and it must be specific to any given physical quantity.

The core of the algorithm (referred to a single sensor node) consists of a read-prediction loop where, at each step, the value of the prediction at time t is com-

puted by the cluster head according to Equation 4.3 using the current values of α and θ.

Meanwhile, each sensor node i in the same cluster keeps collecting samples $z(t)$ of the environmental quantities at its current rate, and sends them toward the cluster head, where they are appended to the set R of the current sensory readings, causing the deletion of the oldest ones.

The accuracy of the prediction model for the specific sensor node is tested by comparing the latest measurement with the corresponding prediction. If the absolute difference is lower than a pre-defined threshold, the sampling rate is increased by doubling the controlling factor ($T \leftarrow 2 \cdot T$); otherwise, the model needs to be adapted to potentially fast variations in the monitored physical phenomenon, so the sampling rate is reset and ($T \leftarrow 1$).

Regardless of the threshold, whenever a sensory reading is received by the cluster head, the transformation parameters, α and θ are updated via a gradient descent algorithm starting from the latest estimates, according to Equation 4.4.

The algorithm 6 reports the steps needed for computing the environmental function $F(x, y, t)$, based on the estimates of the intrinsic parameters of the nodes of the cluster, as shown in Equations 4.1, 4.2, 4.3 and 4.4.

Algorithm 5 Incremental estimate of intrinsic parameters.

Parameters initialization:

$f_{x,y}^{old}(t) \leftarrow \sum_{j=0}^{M-1} w_j \mathcal{N}(t|\mu_j, \sigma_j^2)$ ▷ Δt_{old} Gaussian fitting

$S_r \leftarrow \emptyset, S_t \leftarrow \emptyset$ ▷ daily set of readings and relative time instant

$read\_time \leftarrow t_0$ ▷ beginning of readings

$\alpha \leftarrow 1, \theta \leftarrow 0$ ▷ compression and translation

$\tau \leftarrow \tau^\star, T \leftarrow 1$ ▷ prediction threshold and sensing rate

Read-Prediction cicle:

loop

  while $read\_time > current\_time$ not operate **end while**

  $S_r \leftarrow S_r \cup \{z(t)\}$ ▷ update readings set

  $S_t \leftarrow S_t \cup \{t\}$ ▷ update time set

  $\hat{z}(t) \leftarrow \alpha[f_{x,y}^{old}(t) - \theta] + \theta$ ▷ prediction at time t

  if $|\hat{z}(t) - z(t)| < \tau$ **then** $T \leftarrow 2 * T$ **else** $T \leftarrow 1$

  end if

  $S_p \leftarrow \emptyset$ ▷ initialize prediction set

  for all $t \in S_t$ **do** ▷ compute prediction set

    $S_p \leftarrow S_p \cup \alpha[f_{x,y}^{old}(t_i) - \theta] + \theta$

  end for

  $E_D(\alpha', \theta') = \sum_{z \in S_r, \hat{z} \in S_p} \{z_i - \hat{z}_i\}^2$ ▷ square error of predictions

  $(\alpha, \theta) \leftarrow \underset{\alpha', \theta'}{\mathbf{argmin}}\ E_D$ ▷ gradient descent algorithm

  $read\_time \leftarrow read\_time + T$ ▷ next reading time

end loop

Algorithm 6 Computation of the environmental function $F(x, y, t)$.

Input Parameters:

x_i, y_i ▷ XY coordinates of sensor i

x, y, t ▷ A 3-D point

Computation of $F(x, y, t)$:

for all $i \in [1..N]$ **do** ▷ for all nodes in the cluster

  $d_i = \sqrt{(x - x_i)^2 + (y - y_i)^2}$ ▷ distance from node i

  $w_i = e^{-d_i}$ ▷ mixture weight for node i

end for

$F(x, y, t) = \sum_{i=1}^{N} w_i f_{x_i, y_i}(t) / \sum_{i=1}^{N} w_i$

4.2.2 Estimating the extrinsic parameters

The algorithm for estimating the extrinsic parameters allows to compute a model that may be ported from the reference site to one or more target sites. The basic assumption is that the target sites has at least one real node that links the real nodes to simulated ones.

Figure 4.5 summarizes the required steps for porting the model from a reference site to a single target site: first of all, a list of the spatial coordinates (X_i, Y_i) for all the real nodes lying in the target site is generated. The following step involves the computation of the transformation matrix M that maps the space points (X_i, Y_i) to (x_i, y_i) from the target site to the reference site, as explained in Equation 4.5.

Subsequently, the value of the function $F^{ref}(x_i, y_i, t)$ is computed for each of the mapped points in the reference site. Finally, the procedure described by the Equation 4.7 computes the value of the scaling factor β that fits, as best as possible, the reference site function $F^{ref}(x_i, y_i, t)$ to the values $F^{tar}(X_i, Y_i, T)$ sensed in the target region.

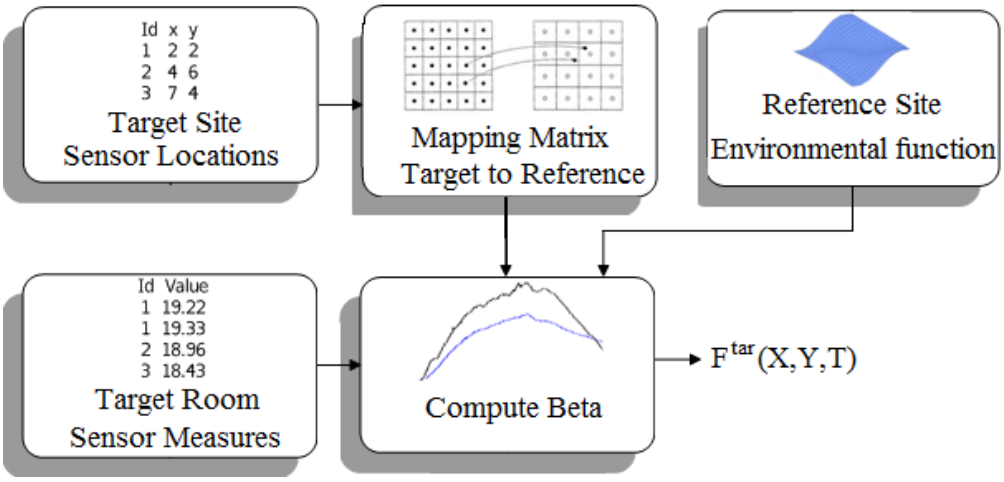


Figure 4.5: Porting of the environmental function to the target environment.

The algorithm 7 formalizes the method devised for porting the model from a reference site to one or more target sites.

It accepts as input J, the number of target sites to which porting the model generated in the reference site, I_j the number of real nodes in the j-th target site, M_{S_j} and $M_{phi,j}$ the scaling and rotation matrix that map the reference site to the j-th target site, and finally $P_{i,j}$ and $S_{i,j}$ that represent respectively the XY location and the readings sensed by node i in the target site j.

The core of the algorithm is a loop that ranges over all the target sites and computes in sequence: the mapping matrix M, its inverse M_{inv}, the points $p_{i,j}$

corresponding to the points $P_{i,j}$ mapped onto the reference site and the readings inferred in the reference site $s_{i,j}(t)$. Finally the value of β_j is computed for each of the target sites as explained in Equation 4.7.

Once the value of the function $F^{tar}(X, Y, T)$ is computed it may be used to simulate as many virtual nodes as needed by the WSN designer or user.

Algorithm 7 Estimate of the beta stretching factors

Input parameters:

J ▷ The number of target sites

I_j ▷ The number of nodes in the j-th target site

$M_{S,j}$ ▷ The j-th scaling matrix

$M_{\phi,j}$ ▷ The j-th rotation matrix

$P_{i,j}$ ▷ XY location of the i-th sensor in the j-th target site

$S_{i,j}(t)$ ▷ The readings sensed by node i in the target site j at time t

Estimation of the beta stretching factors:

loop

  for all $j \in [1..J]$ **do** ▷ Loop on the target sites

    $M = M_{S,j} M_{\phi,j}$ ▷ Reference to target matrix

    $M_{inv} = M^{-1}$ ▷ Target to reference matrix

    for all $i \in [1..I_j]$ **do**

      $p_{i,j} = M_{inv} P_{i,j}$ ▷ The mapped points in the reference site

      $s_{i,j}(t) = F^{ref}(p_{i,j}, t))$ ▷ Inferred readings in the reference site

    end for

    $\beta_j = \underset{\beta}{argmin}\ E(\beta) = \sum_{i \in [1..I_j]} [S_{i,j}(t) - \beta s_{i,j}(t)]^2$

  end for

end loop

Output parameters:

β_j ▷ The scaling factors for all the target sites

Chapter 5

Experimental assessment

The purpose of this Chapter is to evaluate the performance of FDDP whose building blocks (AFD and SDP) have been described in Chapters 3 and 4. The former set of experiments is conducted in absence of data faults and aims at evaluating the capabilities of FDDP to predict data for the light exposure, the temperature and the humidity and to port the environmental functions generated for a reference site to another target site. A brief discussion about the energy saving obtained by using the adaptive sampling provides interesting results and proves the suitability of such an approach to WSNs.

The latter set of experiments considers a more realistic scenario where many sensor nodes gather temperature readings and the data are corrupt by faults. The performances of the FDDP system are thus evaluated with respect to the detection and prediction perspectives.

5.1 Assessment of FDDP in absence of faults

This set of experiments uses real data collected by Mica2-dot sensor nodes contained in the public database available from [49]. The working environment is the Intel Berkeley Research Lab (its size is about $30 \times 40m^2$) containing 54 sensor nodes able to sense three typical physical quantities for sensor networks, namely the temperature, relative humidity, and light exposure.

The measured values range from 17°C to 25°C for the temperature, from 20% to 90% for the relative humidity, and from 0 to 1800 Lux for the light exposure; the observations were carried out over a time period spanning several days, between February, 28 2004 and April, 5 2004; the sampling rate of each node was set to 31s.

The time interval of the experiments spans for nine days from February-29-2004 to March-8-2004; data were manually pre-processed in order to get rid of

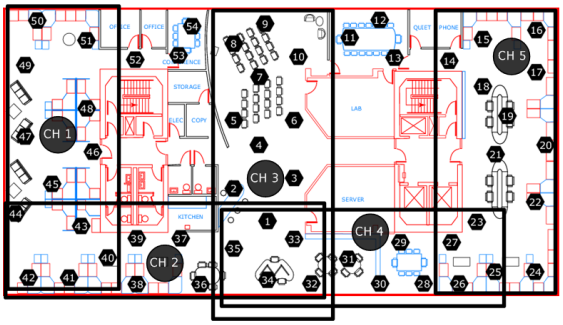


Figure 5.1: Deployment of the clusters for assessing the performance of the prediction module in absence of data faults.

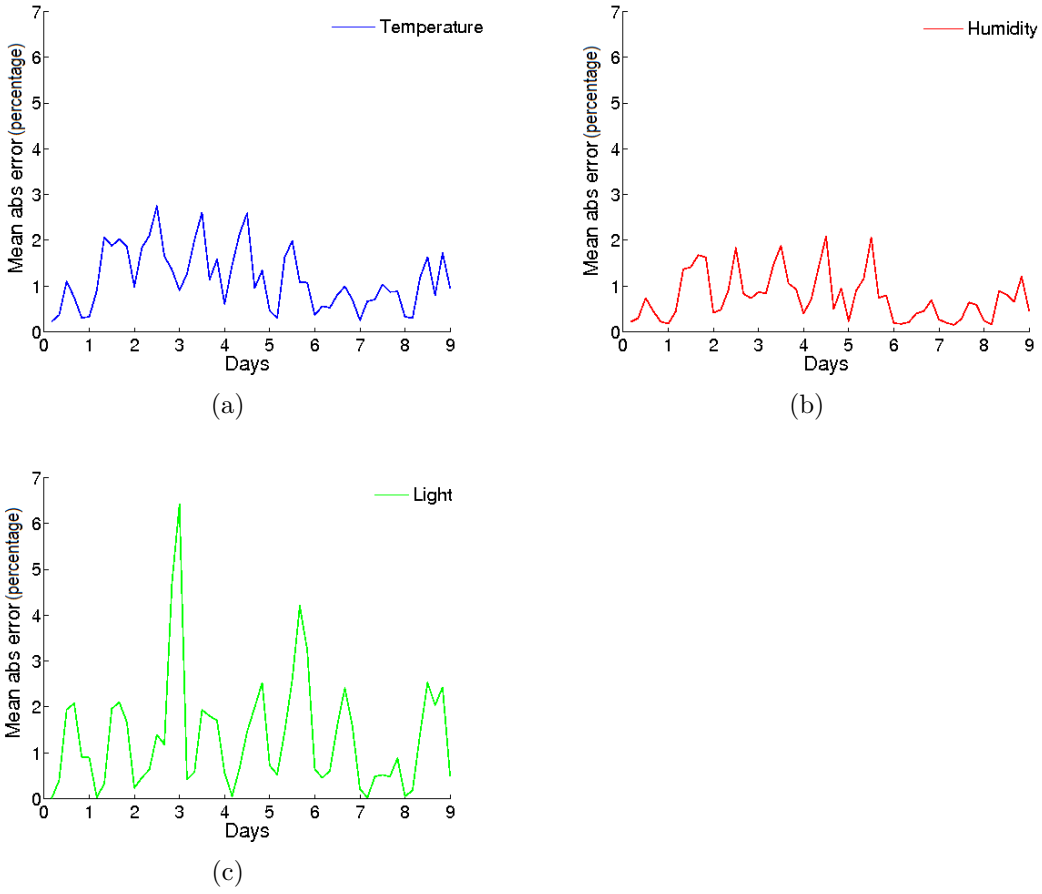


Figure 5.2: Mean absolute error for the quantities predicted in cluster #1.

Table 5.1: Mean Absolute Error (%)

Area	Temperature	Light	Humidity
1	1.1655	1.3488	0.7961
2	1.2774	1.3560	0.7841
3	0.8318	0.6638	0.5042
4	1.3511	1.6086	0.7658
5	1.1695	1.0098	0.6317

Table 5.2: Variance of the Absolute Error ($\%^2$)

Area	Temperature	Light	Humidity
1	0.4523	1.5805	0.2814
2	0.4762	1.1046	0.2273
3	0.3420	0.2408	0.1134
4	0.3441	2.2556	0.1784
5	0.3698	0.5470	0.1255

Table 5.3: Maximum Absolute Error (%)

Area	Temperature	Light	Humidity
1	2.7521	6.4209	3.0874
2	2.9180	4.4249	1.8914
3	2.0053	1.9667	1.6625
4	2.5279	6.4697	1.6638
5	2.4300	2.7350	1.4039

incomplete or clearly erroneous measurements.

The performance of FDDP are assessed in absence of faults and the AFD module is excluded from the evaluation. Different clusters of nearby nodes whose measurements are correlated both in the space and the time dimensions are considered; in particular, an accurate study reported in [50] identified five areas, where the measured quantities show high spatial correlation, and the same layout is used for the experiments discussed in this Section. SDP is thus run in the hypothetical scenario where each area is managed by a cluster head directly connected to each of the original nodes as summarized in Figure 5.1.

The performance metrics (mean, variance and maximum absolute prediction error) were normalized with respect to the previously mentioned ranges in order to

obtain uniform comparison of the results obtained for different physical quantities.

Each of the cluster heads depicted in Figure 5.1 is assumed to run the Algorithm 5 for estimating the intrinsic parameters of the managed nodes; the threshold used to invalidate the model depending on the tolerable prediction error is set to 5%, and the periodicity of each physical quantity is assumed to be 24 hours.

The reliability of the produced predictive models has been assessed by computing the mean absolute error for each area and for each physical quantity; in particular Figure 5.2 plots the the mean absolute prediction error for the test area #1 for temperature, humidity, and light, respectively; other areas show analogous trends.

The values of the mean absolute error for all areas are reported in Table 5.1, while Table 5.2 reports the related variance and Table 5.3 maximum values. It is worth noting that the mean prediction error is very low in all cases (about 1%); in addition the largest variance is observed for light in area 4, but it is still not very high (2.25%); moreover, peaks of error only occur occasionally and remain localized to very few time instants, with the maximum amounting to ∼6% for light in areas 1 and 4.

5.1.1 Energy saving assessment

The number of samples needed by the algorithm for estimating the intrinsic parameters rescales the sampling period of the sensor nodes based on the accuracy of the current predictor. A good indicator that allows to evaluate its performance in terms of energy saving, is the number of the overall required sensing and transmissions across all nodes compared to the original set of raw data.

In order to avoid large prediction errors the sampling rates are upper-bounded to 8 minutes, whereas the basic sampling rate is slightly over 30 seconds; this results into a sampling frequency ranging from a minimum value of $2,1 \cdot 10^{-3} Hz$ up to a maximum of $33,3 \cdot 10^{-3} Hz$.

Figure 5.3 plots the mean sampling frequency for area 1, for all of the considered quantities, and Table 5.4 and Table 5.5 show mean and variance of sampling frequency for each area, respectively.

The largest values for the sampling rate (i.e. $4.31 \cdot 10^{-3} Hz$) occur for light, due to the intrinsic nature of such physical quantity, which exhibits lower autocorrelation than temperature and humidity. Nevertheless, such value represents a significant improvement as compared to the basic sampling rate of $33,3 \cdot 10^{-3} Hz$, meaning that, on average, only ∼12% samples are needed with respect to the maximum frequency. Temperature and humidity sampling rates are quite close to the minimum sampling rate of $2.1 \cdot 10^{-3} Hz$, thanks to the higher autocorrelation characteristic of these quantities.

In order to quantify the impact of the proposed approach in terms of energy saving, we compared the number of necessary transmissions with respect to the basic approach where each node plainly forwards its samples to the collecting station at the basic, fixed sampling rate.

Table 5.6 shows the percentage of measurements actually needed by our algorithm in order to produce models that are deemed reliable, with respect to the thresholds defined for the prediction error.

We note that the number of samples needed in the worst case amounts to 8.5% for temperature, 6.8% for relative humidity and 12% for light, which is the environmental quantities with higher variance. Energy saving on transmission and sensing is thus greater than 88% in all cases.

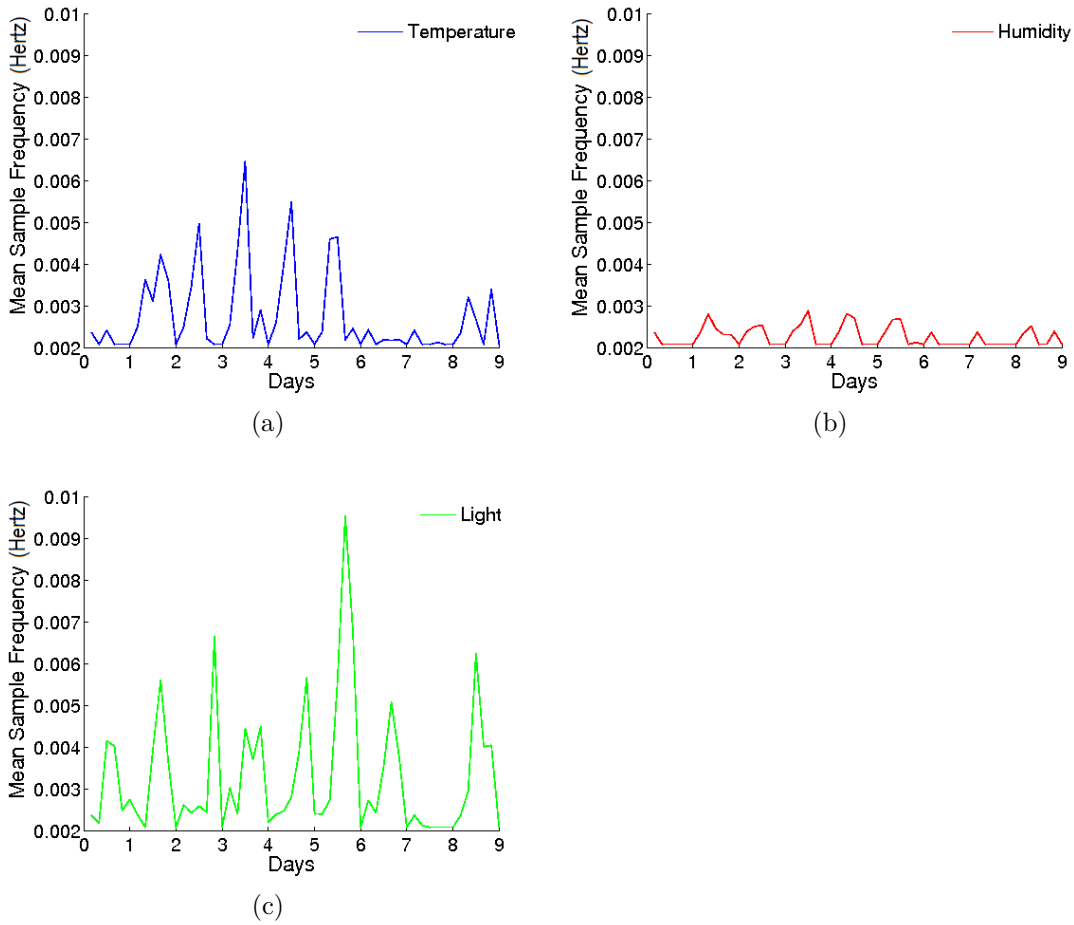


Figure 5.3: Mean sampling frequency for the nodes of area #1.

Table 5.4: Mean Sampling Rate ($Hz \cdot 10^{-3}$)

Area	Temperature	Light	Humidity
1	2.8120	3.3790	2.3340
2	2.9020	3.5690	2.3450
3	2.4870	2.4760	2.2910
4	2.8650	4.3120	2.2990
5	2.7100	3.1110	2.2660

Table 5.5: Variance of the Sampling Rate ($Hz^2 \cdot 10^{-6}$)

Area	Temperature	Light	Humidity
1	1.0220	2.4120	0.0580
2	1.2170	2.8990	0.0570
3	0.1990	0.2050	0.0210
4	0.5040	9.1260	0.0380
5	0.3900	1.6400	0.0280

Table 5.6: Fraction of Samples Used (%)

Area	Temperature	Light	Humidity
1	8.1602	9.8903	6.7943
2	8.5154	10.2800	6.7686
3	7.2849	7.2331	6.5291
4	8.2898	12.7218	6.6528
5	7.9044	9.2566	6.6027

5.1.2 Modeling and porting assessment

The Algorithms 6 and 7 respectively build the environmental functions of the monitored quantity for the reference and target site. This set of experiments aims at assessing the precision of the inferred prediction models for two sub-areas of the Berkeley Intel Laboratory, and in particular for the sensor nodes deployed in Figure 5.4.

Based on the groups of sensor nodes highlighted in the figure, “Site 1” is the reference site and “Site 2” is the target site; experimental results shows that sensory readings can be reliably simulated for the reference site and that the ported models are reliable for the target site.

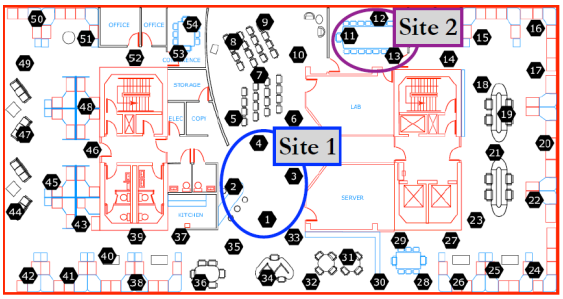


Figure 5.4: Deployment of the reference and target site for evaluating the porting capabilities of FDDP.

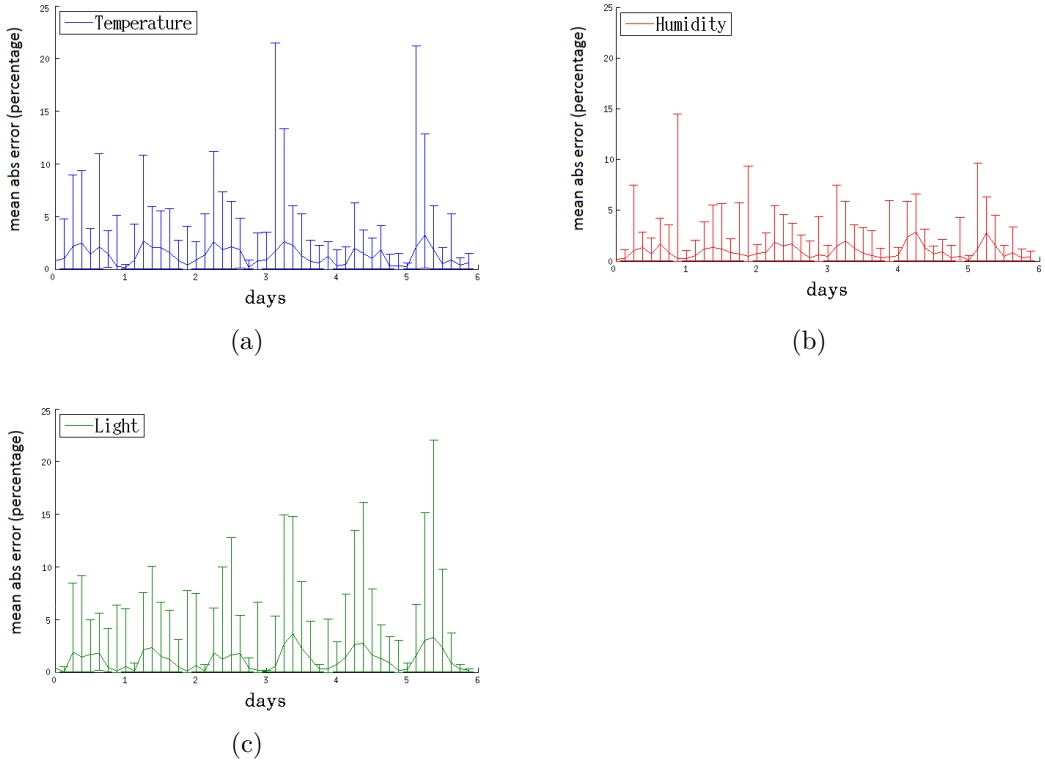


Figure 5.5: Mean Prediction error and variance of the error for the nodes of the reference site.

The target site has few actually deployed nodes; in particular, “Site 2” contains nodes 11, 12 and 13, but only node 11 is considered for training, whereas nodes 12, and 13 are used for testing the precision of the ported models; hence, by using the readings from node 11 as probes, the models from “Site 1” can be adapted to the

new environment, following the method described in Section 4.1.2 for computing the extrinsic parameters.

Figure 5.5 shows the mean absolute error and the variance of the error for the nodes of the reference site. In particular all available nodes in Site 1 (nodes 1, 2, 3, 4) were used to perform some tests in a leave-one-out fashion, i.e. the models are built using data from N -1 nodes while the prediction error is computed on

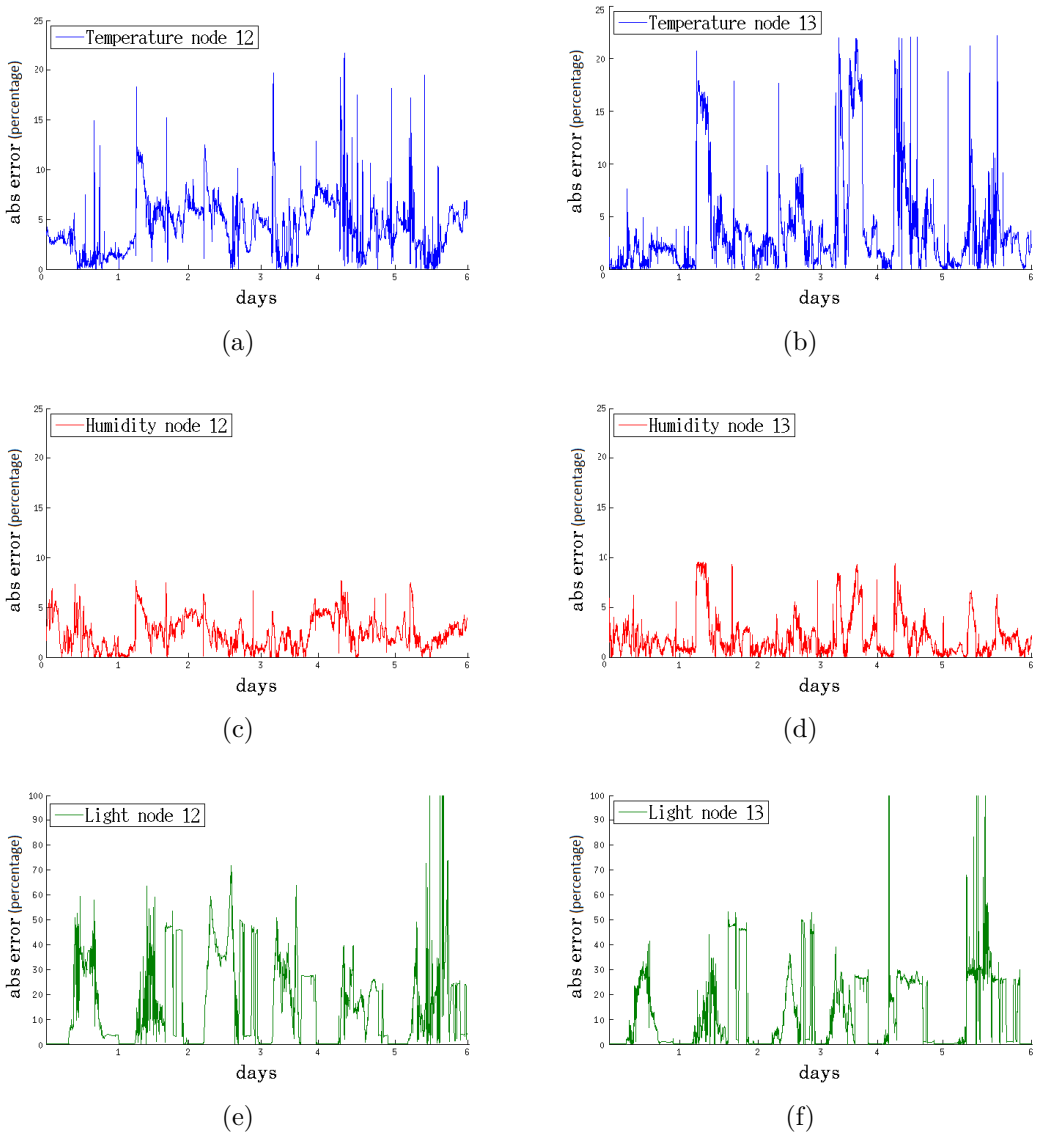


Figure 5.6: Absolute prediction error for node 12 and 13 when porting models from the reference to the target site.

the readings from the Nth node. The tests were thus conducted using 3 nodes for training and 1 for testing; experiments run from February-28-2004 to March-5-2004 and the first day is used for training. The absolute error (averaged across all runs of leave-one-out) is 0.0238 for temperature, 0.0178 for humidity, and 0.0285 for light.

Figure 5.6 shows the prediction errors for all the considered quantities, for both the testing nodes of the target site. The errors are still quite acceptable for all the three quantities, thus proving that models computed on "Site 1" and adapted to "Site 2" by using probes from just one node provide reliable estimates in the two considered test points. It is also worth noting that while errors relative to temperature and humidity measurements are below 25%, and 10% respectively, errors for light measurements are considerably higher; this is due to intrinsically lower correlation for Light, which makes it harder to port the model from one site to another.

5.2 Assessment of FDDP in presence of faults

This set of experiments evaluates both the detection and the prediction algorithms when the data are affected by fault.

The WSN was arranged as tree characterized by depth 16 and branch factor 3, and composed by 100 sensor nodes gathering temperature measurements every 30 seconds for two days.

The readings coming from ten Mica2Dot were recorded in a basic dataset and used as real generators; in a later stage, the dataset was modified by adding nine different Gaussian noise signals $\mathcal{N}(0, \sigma_E^2)$, with $\sigma_E^2 = 0.02$, to the original sensory signal, thus producing a dataset of 100 among real and virtual sensor nodes. Data faults have been simulated by corrupting the obtained dataset, according to fault definitions proposed in [44]:

- *Outlier:* $\tilde{r}(t) = g \times r(t)$
- *Noise:* $\tilde{r}(t) = r(t) + \mathcal{N}(0, \sigma_N^2)$
- *Stuck-at:* $\tilde{r}(t_0 + i) = r(t_0) + \mathcal{N}(0, \sigma_\theta^2), i \in \{1, .., k\}$,

where $g = 1.5$ is a gain constant, $\sigma_N^2 = 3$ and $\sigma_\theta^2 = 10^{-9}$ are Gaussian noises, t_0 is a random time instant where a constant fault starts and $K = 10$ is its duration. Different scenario dynamics have been simulated by generating three different corrupt datasets where the amount of data faults corresponds respectively to 20%, 30% and 40% of total number of readings. In each dataset, different classes of faults were mixed in equal parts. For each dataset, the first day of readings

was used as training set to learn conditional probability tables for the Bayesian network, while the second day was used as the test set.

5.2.1 Fault detection assessment

The performance of AFD (Algorithm 2), with different QoS constraints over the response time and the classification accuracy, were compared against two benchmarks corresponding to the max-product inference algorithm over two different static network topologies.

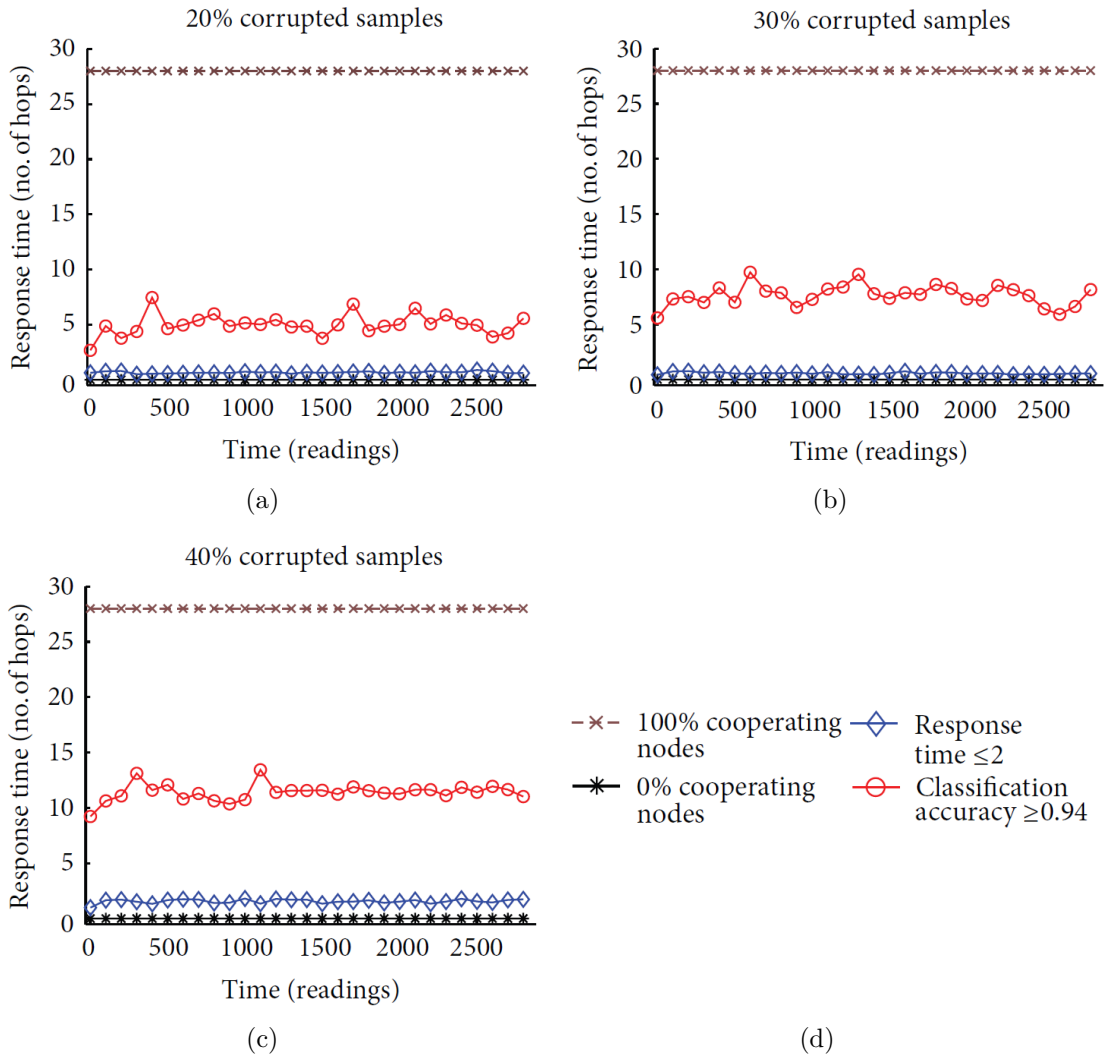


Figure 5.7: Response time for three different amount of corruption and four network topologies.

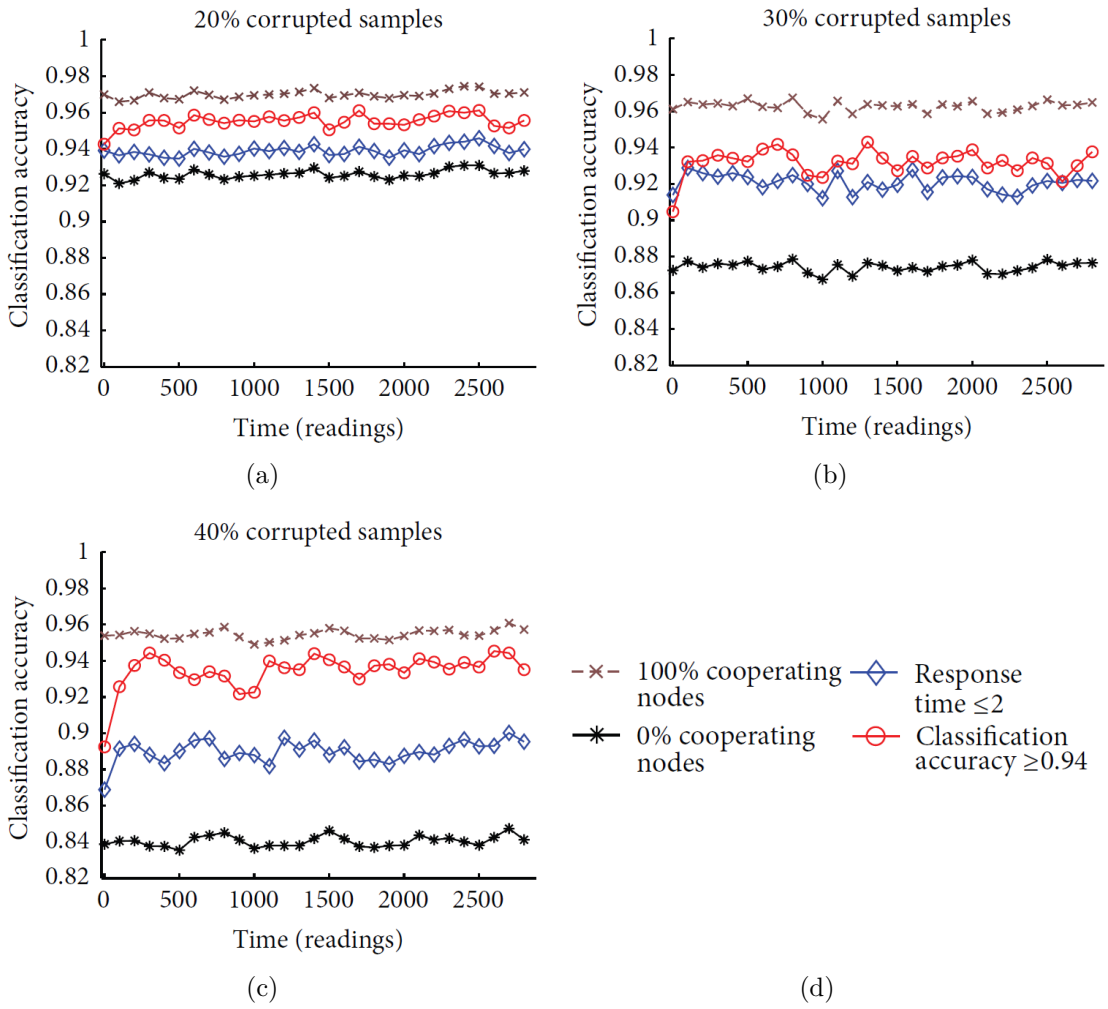


Figure 5.8: Classification rate for three different amount of corruption and four network topologies.

The first benchmark corresponds to a fully connected network, i.e. a single cluster where 100% of nodes cooperate: the average response time is proportional to the number of hops required by the algorithm that are fixed to 28 hop; the classification accuracy also achieves its upper bound.

The second benchmark corresponds to a network where none of the nodes cooperate, i.e. the number of clusters is equal to the number of nodes; in this benchmark the average response time is equal to 0 and the classification accuracy drops to its lower bound. We want to show that the adaptive behavior of AFD makes it able to meet the imposed constraints, while paying a small cost in terms of the possibly unconstrained metric; at the same time, although the static topologies

achieve better performance with respect to one QoS requirement, they dramatically worsen the other one. We adopted two different set of QoS constraints, namely:

$$v_1 = (v_{err} = 0.06, v_{lat} = unconstrained),$$
$$v_2 = (v_{err} = unconstrained, v_{lat} = 2). \tag{5.1}$$

Sensor nodes run the Pareto optimization every $T = 100$ readings.

Performances are evaluated by using the two considered QoS metrics, namely the classification accuracy (a value between 0 and 1) and the response time (measured in numbers of hops), averaged over time windows of size T and over all the networks nodes.

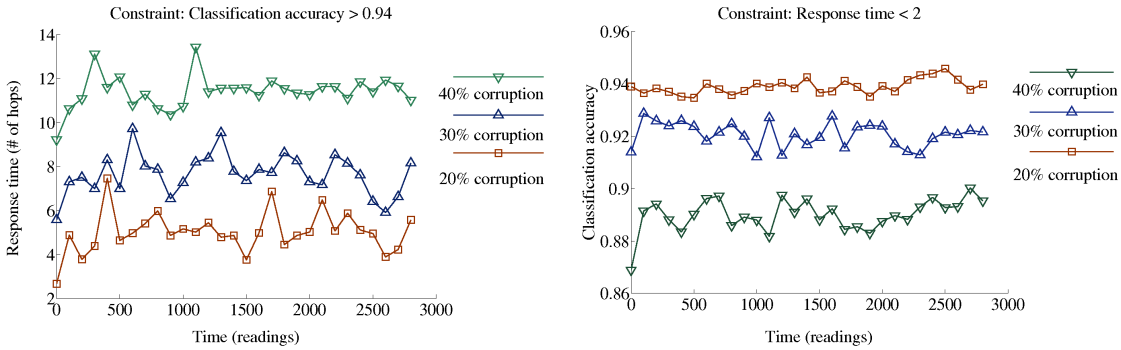
Figures 5.7 and 5.8 compare the performances of the four considered configurations in three different scenarios. As expected, the "100% cooperating node" configuration always achieves the best classification accuracy and the worst response time in every scenario, while the "0% cooperating node" configuration is the best for response time and the worst for the classification accuracy.

Analyzing the performance of AFD with a constraint over the classification accuracy (v_1), it is possible to note that such metric approaches the constraint for every scenario, paying an increasing cost in response time as the corruption percentage increases. Moreover, while the classification accuracy is quite close to the "100% benchmark", in comparison, its response time obtains a significant reduction.

Analogously, AFD with a constraint over the response time (v_2) always meets such requirement by paying a small decrease in the classification accuracy as the corruption percentage increases. Moreover, at the cost of a small increase of response time with respect to the "0% benchmark", the classification accuracy is remarkably higher.

Experimental results demonstrate that AFD is able to adapt its behavior to different scenarios, i.e. to different percentages of corruption in sensory data. This adaptability corresponds to tuning the unconstrained QoS metrics in order to satisfy the constrained ones. Figure 5.9 shows this property by comparing the performance of AFD, respectively with constraints v_1 and v_2, for the three considered scenarios. In particular, Figure 5.9 (a) shows how AFD behaves when a constraint over the classification accuracy is imposed; as shown, it rapidly adapts the network communication graph, thus increasing the response time for higher percentage of corruption. Analogously, Figure 5.9 (b) shows that, as the amount of corrupt samples increases, AFD with a constraint over the response time is able to tune the classification accuracy by decreasing it after few time steps.

Such results show that AFD is able to find the correct trade-off among the QoS metrics, satisfying the imposed constraints and taking implicitly into account also the dynamics of the real deployment scenario.



(a) (b)

Figure 5.9: A comparison among the average classification accuracy and response times for different constraints.

Finally, experimental results allow also to evaluate qualitatively the relationship between response time and classification accuracy in three different scenarios, as shown in Figure 5.10. The plots shown were obtained by averaging values obtained in the four settings described in previous experiments.

Trends are quite similar for all the three scenarios: the minimum value of the classification accuracy is clearly achieved for singleton clusters; on the contrary, a fully connected network gets the maximum of the classification accuracy.

A small increase of the allowed response time, for small cluster size, corresponds to great increase of the classification accuracy. This behavior is more visible when the corruption amounts to 20%; in this scenario a response time proportional to 5

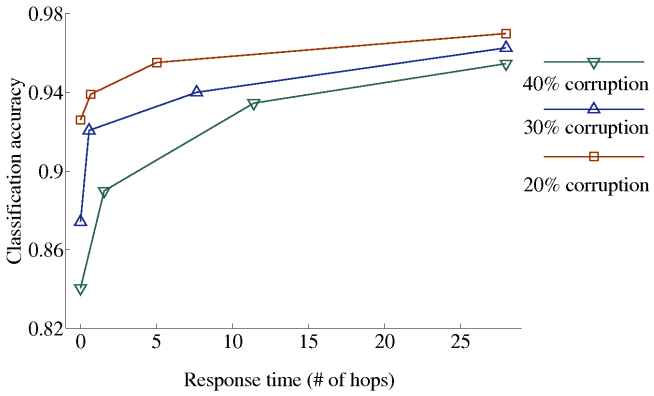


Figure 5.10: Average response time and accuracy for the three considered scenarios.

hops corresponds to a classification accuracy very close to its maximum value.

The three plots tend to asymptotically reach their maximum value as the response time increases. Such evidence allows to deduce that farther nodes have a small influence on the choice of the correct class label; this behavior is strictly related to the characteristic of the monitored physical quantity (e.g. the temperature), and simply means that spatial correlation decreases as the distance between nodes increases.

5.2.2 Prediction and energy saving assessment

The performance of SDP are now reevaluated in a context where data are affected by errors but after the action of the AFD module.

The mean absolute prediction error is measured in percentage and normalized within the range of temperature 18°C to 35°C, while the sampling rates are expressed in percentage of the total amount of samples of the starter dataset (30 seconds sampling rates temperature data). All the results are averaged over the 100 sensor nodes and over the four network topologies mentioned previously.

Figure 5.11 highlights that the mean absolute error is quite limited (under 5%) thanks to the filtering effect provided by AFD. The error in the three considered scenarios is comparable to the results obtained in absence of faults for the previous experiments (see Table 5.4). It is quite interesting noting that the best performance are achieved in the scenario where the corruption amounts to 40%: basically, whenever a node recognizes that the gathered reading is corrupt, its sampling rate is reset to its maximum and the node is forced to keep on sensing until it produces meaningful readings. As a result, the related predictor would exhibit a high precision at the cost of an increased sampling rate.

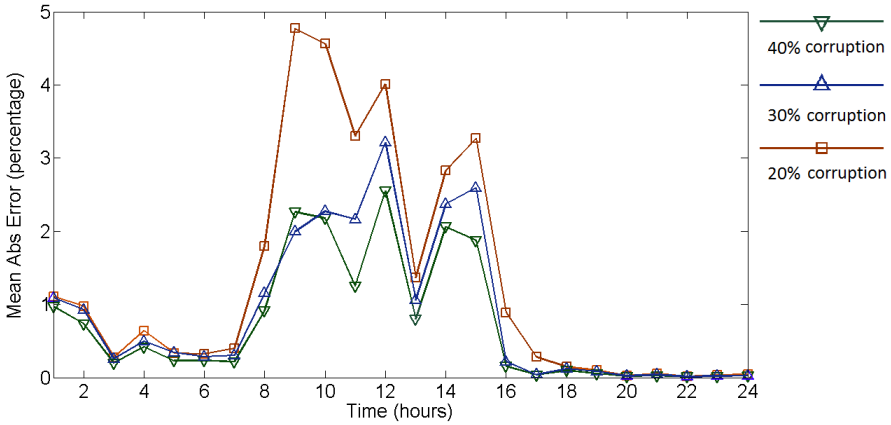


Figure 5.11: Mean prediction error for the three simulated scenarios.

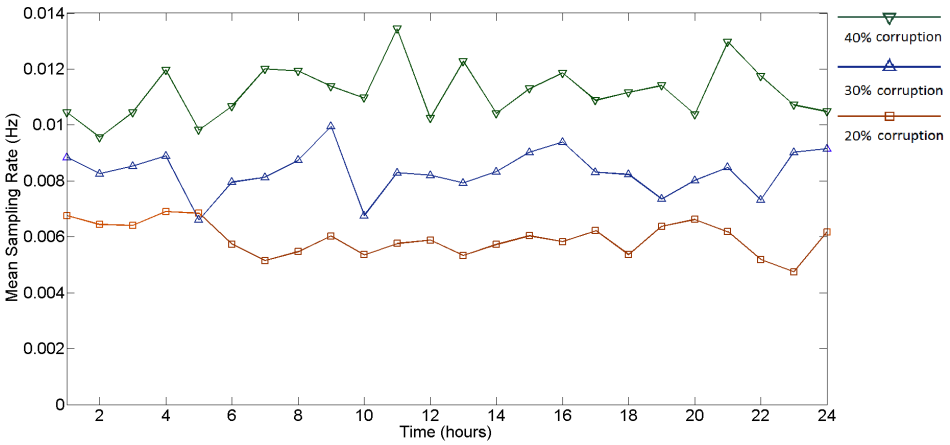


Figure 5.12: Mean sampling rates for the three simulated scenarios.

Figure 5.12 shows the sampling rates for the scenarios considered. As expected the highest sampling rate is required in the case of 40% of total corruption and amounts to 0.011 Hertz on average; however, such a value is still quite far from to the maximum sampling rate of 0.0333 Hertz (30 seconds) and corresponds about to a 67% of energy saving. The 20% scenario required instead a sampling rate of 0.006 Hertz on average and saved about the 87% of energy with respect to the maximum sampling rate.

Conclusions

This dissertation described FDDP, a system which builds reliable prediction models for the data generated by a WSN. The developed system was able to predict common physical phenomena namely the temperature, the humidity and the light exposure by means of the SDP software module. Experimental results demonstrated the ability of FDDP to make reliable prediction for the real network nodes as well as to generalize the models to unknown environments. The use of prediction models allowed to reduce the energy consumption of sensor nodes by adaptively tuning their sampling rate. Whenever the difference between the measurements and the prediction was within a specific treshold the sampling rate of the node was lowered. This allowed to reduce sensing and communication toward the base station that resulted in an overall reduction of the energy consumption of sensor nodes.

When the data are affected by faults, the prediction models may deviate substantially from the reality and this issue was addressed by the AFD module. It implements a cooperative distributed algorithm and uses a Bayesian networks to detect corrupt measurements within the WSN. Theoretical results have shown that unlike other works in literature, the algorithm scales well in time, message and memory complexity. A large set of experiments using data gathered from a real scenario demonstrated the benefits provided by the AFD module for filtering out corrupt readings; in particular, this allowed FDDP to produce prediction models with high precision properties also when the amount of corrupt data was very high, only at the cost of a little increase in the sampling rate of the network nodes.

Bibliography

[1] Ed Callaway, Paul Gorday, Lance Hester, Jose A Gutierrez, Marco Naeve, Bob Heile, and Venkat Bahl. Home Networking with IEEE 802.15. 4: a Developing Standard for Low-Rate Wireless Personal Area Networks. *IEEE Communications Magazine*, 40(8):70–77, 2002.

[2] Jianping Song, Song Han, Aloysius K Mok, Deji Chen, Mike Lucas, and Mark Nixon. WirelessHART: Applying Wireless Technology in Real-Time Industrial Process Control. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'08)*, pages 377–386. IEEE, 2008.

[3] Patrick Kinney. ZigBee Technology: Wireless Control that Simply Works. In *Communications design conference*, volume 2. ZigBee Alliance, 2003.

[4] Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, and David Culler. TinyOs: An Operating System for Sensor Networks. *Ambient Intelligence*, 35, 2005.

[5] Deborah Estrin, Lewis Girod, Greg Pottie, and Mani Srivastava. Instrumenting the World with Wireless Sensor Networks. In *Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2001)*, volume 4, pages 2033–2036. IEEE, 2001.

[6] Luca Benini, Elisabetta Farella, and Carlotta Guiducci. Wireless Sensor Networks: Enabling Technology for Ambient Intelligence. *Microelectronics Journal*, 37(12):1639–1649, 2006.

[7] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless Sensor Networks: a Survey. *Computer networks*, 38(4):393–422, 2002.

[8] Esteban Egea-Lopez, Javier Vales-Alonso, Alejandro S Martinez-Sala, Pablo Pavon-Marino, and Joang García-Haro. Simulation Tools for Wireless Sensor Networks. In *Proceedings of the International Symposium on Performance*

Evaluation of Computer and Telecommunication Systems (SPECTS05), page 24, 2005.

[9] Marko Korkalainen, Mikko Sallinen, Niilo Kärkkäinen, and Pirkka Tukeva. Survey of Wireless Sensor Networks Simulation Tools for Demanding Applications. In *Fifth International Conference on Networking and Services, 2009. ICNS '09.*, pages 102–106. IEEE, 2009.

[10] Arran Holmes, Hakan Duman, and Anthony Pounds-Cornish. The iDorm: Gateway to Heterogeneous Networking Environments. In *International ITEA Workshop on Virtual Home Environments*, pages 30–37. VHE Middleware Consortium, 2002.

[11] Victor Lesser, Michael Atighetchi, Brett Benyo, Bryan Horling, Victor Lesser Michael Atighetchi, Anita Raja, Regis Vincent, Ping Xuan, Shelley XQ. Zhang, Thomas Wagner, Ping Xuan, and Shelley Xq. Zhang. The intelligent home testbed. In *Proceedings of the Autonomy Control Software Workshop*, 1999.

[12] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In *Proceedings of the 1st International Conference on Embedded networked Sensor Systems*, pages 126–137. ACM, 2003.

[13] Olaf Landsiedel, Klaus Wehrle, and Stefan Götz. Accurate Prediction of Power Consumption in Sensor Networks. In *The Second IEEE Workshop on Embedded Networked Sensors, 2005. EmNetS-II.*, pages 37–44. IEEE, 2005.

[14] E-O Blass, Jens Horneber, and Martina Zitterbart. Analyzing Data Prediction in Wireless Sensor Networks. In *IEEE Vehicular Technology Conference, 2008 (VTC 2008)*, pages 86–87. IEEE, 2008.

[15] Raja Jurdak, X Rosalind Wang, Oliver Obst, and Philip Valencia. Wireless Sensor Network Anomalies: Diagnosis and Detection Strategies. In *Intelligence-Based Systems Engineering*, volume 10 of *Intelligent Systems Reference Library*, pages 309–325. Springer Berlin Heidelberg, 2011.

[16] Salvatore Gaglio, Luca Gatani, Giuseppe Lo Re, and Alfonso Urso. A logical Architecture for Active Network Management. *Journal of Network and Systems Management*, 14(1):127–146, 2006.

[17] Alessandra De Paola, Salvatore Fiduccia, Salvatore Gaglio, Luca Gatani, Giuseppe Lo Re, A Pizzitola, Marco Ortolani, Pietro Storniolo, and Alfonso Urso. Rule Based Reasoning for Network Management. In *Proceedings of*

the Seventh International Workshop on Computer Architecture for Machine Perception (CAMP 2005), pages 25–30. IEEE, 2005.

[18] Alessandra De Paola, Salvatore Gaglio, Giuseppe Lo Re, and Marco Ortolani. Sensor9k: A testbed for designing and experimenting with WSN-based ambient intelligence applications. *Pervasive and Mobile Computing*, 8(3):448–466, 2012.

[19] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly Detection: A Survey. *ACM Computing Surveys*, 41(3):15:1–15:58, 2009.

[20] Kevin Ni, Nithya Ramanathan, Mohamed Nabil Hajj Chehade, Laura Balzano, Sheela Nair, Sadaf Zahedi, Eddie Kohler, Greg Pottie, Mark Hansen, and Mani Srivastava. Sensor Network Data Fault Types. *ACM Transactions on Sensor Networks*, 5(3), 2009.

[21] Farinaz Koushanfar, Miodrag Potkonjak, and Alberto Sangiovanni-Vincentelli. On-Line Fault Detection of Sensor Measurements. In *Proceedings of IEEE Sensors, 2003*, volume 2, pages 974–979. IEEE, 2003.

[22] Kevin Ni and Greg Pottie. Bayesian Selection of Non-Faulty Sensors. In *IEEE International Symposium on Information Theory, 2007*, pages 616–620. IEEE, 2007.

[23] Jinran Chen, Shubha Kher, and Arun Somani. Distributed Fault Detection of Wireless Sensor Networks. In *Proceedings of the 2006 Workshop on Dependability issues in Wireless Ad Hoc Networks and Sensor Networks (DIWANS '06)*, pages 65–72. ACM, 2006.

[24] Weili Wu, Xiuzhen Cheng, Min Ding, Kai Xing, Fang Liu, and Ping Deng. Localized Outlying and Boundary Data Detection in Sensor Networks. *IEEE Transactions on Knowledge and Data Engineering*, 19(8):1145–1157, 2007.

[25] Sutharshan Rajasegarar, Christopher Leckie, Marimuthu Palaniswami, and James C Bezdek. Distributed Anomaly Detection in Wireless Sensor Networks. In *Proceedings of the 10th IEEE Singapore International Conference on Communication systems (ICCS 2006)*, pages 1–5, 2006.

[26] X Rosalind Wang, Joseph T Lizier, Oliver Obst, Mikhail Prokopenko, and Peter Wang. Spatiotemporal Anomaly Detection in Gas Monitoring Sensor Networks. In *Wireless Sensor Networks*, volume 4913 of *Lecture Notes in Computer Science*, pages 90–105. Springer Berlin Heidelberg, 2008.

[27] Peng Yang, Qingsheng Zhu, and Xun Zhong. Subtractive Clustering Based RBF Neural Network Model for Outlier Detection. *Journal of Computers*, 4(8):755–762, 2009.

[28] Carlos M Fonseca and Peter J Fleming. Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. In *Proceedings of the Fifth International Conference On Genetic Algorithms*, volume 93, pages 415–423, 1993.

[29] Rob Hoes, Twan Basten, Chen-Khong Tham, Marc Geilen, and Henk Corporaal. Quality-of-Service Trade-Off Analysis for Wireless Sensor Networks. *Performance Evaluation*, 66(3-5):191–208, 2009.

[30] Ho Ting Cheng and Weihua Zhuang. Pareto Optimal Resource Management for Wireless Mesh Networks with QoS Assurance: Joint Node Clustering and Subcarrier Allocation. *IEEE Transactions on Wireless Communications*, 8(3):1573–1583, 2009.

[31] Shafaq B Chaudhry, Victor C Hung, Ratan K Guha, and Kenneth O Stanley. Pareto-Based Evolutionary Computational Approach for Wireless Sensor Placement. *Engineering Applications of Artificial Intelligence*, 24(3):409 – 425, 2011.

[32] Alessandra De Paola, Salvatore Gaglio, Giuseppe Lo Re, and Marco Ortolani. Multi-Sensor Fusion through Adaptive Bayesian Networks. In *AI*IA 2011: Artificial Intelligence Around Man and Beyond*, volume 6934 of *Lecture Notes in Computer Science*, pages 360–371. Springer Berlin Heidelberg, 2011.

[33] Giuseppe Anastasi, Marco Conti, Mario Di Francesco, and Andrea Passarella. Energy Conservation in Wireless Sensor Networks: A Survey. *Ad Hoc Networks*, 7(3):537 – 568, 2009.

[34] Wan Du, David Navarro, Fabien Mieyeville, and Frédéric Gaffiot. Towards a Taxonomy of Simulation Tools for Wireless Sensor Networks. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010.

[35] Teerawat Issariyakul and Ekram Hossain. *An Introduction to Network Simulator NS2*. Springer, 2012.

[36] András Varga and Pázmány Péter Sétány. The OMNeT++ Discrete Event Simulation System. In *Proceedings of the European Simulation Multiconference (ESM 2001)*, volume 9, page 185, 2001.

[37] Ankur Jain, Edward Y Chang, and Yuan-Fang Wang. Adaptive Stream Resource Management Using Kalman Filters. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 11–22. ACM, 2004.

[38] Rudolph Emil Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[39] Reynold Cheng, Dmitri V Kalashnikov, and Sunil Prabhakar. Evaluating Probabilistic Queries Over Imprecise Data. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 551–562. ACM, 2003.

[40] Samir Goel, Andrea Passarella, and Tomasz Imielinski. Using Buddies to Live Longer in a Boring World [Sensor Network Protocol]. In *Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops 2006)*. IEEE, 2006.

[41] Daniela Tulone and Samuel Madden. PAQ: Time Series Forecasting for Approximate Query Answering in Sensor Networks. In *Wireless Sensor Networks*, pages 21–37. Springer, 2006.

[42] Mohsen Mollanoori, M Majid Hormati, and Nasrollah M Charkari. An Online Prediction Framework for Sensor Networks. In *16th Iranian Conference on Electrical Engineering*, 2008.

[43] Harry Zhang. The Optimality of Naive Bayes. In *Proceedings of the 17th International Florida Artificial Intelligence Research Society Conference*, pages 562–567, 2004.

[44] Abhishek B Sharma, Leana Golubchik, and Ramesh Govindan. Sensor Faults: Detection Methods and Prevalence in Real-World Datasets. *ACM Transactions on Sensor Networks*, 6(3):23:1–23:39, 2010.

[45] Yair Weiss and William T Freeman. On the Optimality of Solutions of the Max-Product Belief-Propagation Algorithm in Arbitrary Graphs. *IEEE Transactions on Information Theory*, 47(2):736–744, 2001.

[46] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and Tamt Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[47] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[48] Antonio Lalomia, Giuseppe Lo Re, and Marco Ortolani. A Hybrid Framework for Soft Real-Time WSN Simulation. In *Proceedings of the 2009 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications*, pages 201–207. IEEE, 2009.

[49] Samuel Madden. Intel Lab Data. `http://db.csail.mit.edu/labdata/labdata.html`, 2004.

[50] Carlos Guestrin, Peter Bodik, Romain Thibaux, Mark Paskin, and Samuel Madden. Distributed Regression: an Efficient Framework for Modeling Sensor Network Data. In *Third International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 1–10. IEEE, 2004.