Diss. ETH Nr. 18668

Adaptive Sensor Selection Algorithms for Wireless Sensor Networks

A dissertation submitted to
ETH ZURICH

for the degree of
Doctor of Sciences

presented by
Silvia Santini
Dottore in Ingegneria delle Telecomunicazioni
University of Rome "La Sapienza"
born November 15, 1978
citizen of Italy

accepted on the recommendation of
Prof. Dr. Friedemann Mattern, examiner
Prof. Dr. Wendi Heinzelman, co-examiner
Prof. Dr. Kay Römer, co-examiner

2009

Abstract

A wireless sensor network is a collection of tiny, autonomously powered devices – commonly called sensor nodes – endowed with sensing, communication, and processing capabilities. Once deployed over a region of interest, sensor nodes can collect fine-grained measurements of physical variables, like the temperature of a glacier or the concentration of a pollutant. To report their readings to one or more data sinks, sensor nodes communicate using their integrated radio-transceivers and build ad-hoc – possibly multi-hop – relay networks. Thanks to the potentially large number of nodes they are composed of and their ability to operate unattended for long periods of time, wireless sensor networks allow monitoring the environment at an unprecedented spatial and temporal scale.

However, enabling a wireless sensor network to reliably report large quantities of data over long periods of time is still a challenging goal. In particular, since the operation of the radio is known to be the major factor of energy consumption on sensor nodes, limiting communication is crucial for increasing the lifetime of the network. On the other hand, meeting the requirements of wireless sensor network applications may require sensor nodes to collect and report large amounts of sensor readings. The efficient operation of a wireless sensor network thus requires careful scheduling of node participation in sensing and communication. Beyond the role that medium access control and routing protocols may play in this context, so-called sensor selection algorithms can provide for significant communication savings by identifying subsets of the deployed nodes that are sufficient to comply with the application requirements.

This thesis argues for endowing sensor selection algorithms with the ability to dynamically adapt to the observed data and to the local topology of the network. The presented work offers novel sensor selection strategies that can continuously tune their parameters in a distributed fashion, thereby relying on no or only little a priori knowledge about the phenomena of interest. In particular, the thesis first addresses the

sensor selection problem in the time domain by considering scenarios in which sensor nodes autonomously adapt their data reporting rate. To this end, nodes report their sensor readings along with a forecasting model, which the sink can in turn use to estimate future readings. Nodes can thus suppress data communication as long as the estimation error at the sink does not grow beyond a pre-specified, application-dependent threshold. Since the choice of a proper forecasting model is instrumental in allowing for high communication savings, this thesis proposes a generic and lightweight procedure to perform adaptive model selection on sensor nodes. The proposed algorithm concurrently maintains and evaluates the performance of a set of models on the nodes and lets them report to the sink the model with the lowest expected communication overhead. Second, the thesis addresses the sensor selection problem in the spatial domain, in particular for applications requiring spatial coverage of a region of interest. It proposes a novel sensor ranking strategy to efficiently schedule the activation of sensor nodes across the deployment region. Information about the local network topology is used to determine the actual relevance of a node for the current sensing task. The resulting ranking of the nodes, properly combined with randomization techniques, is then used to select subsets of nodes whose activation can guarantee compliance with specific application requirements.

The sensor selection strategies proposed in this thesis require little memory and computational resources, and are thus implementable on resource-constrained wireless sensor network platforms. Their practical feasibility is evaluated by means of simulations, experiments on a small-scale deployment, and in the context of a concrete application scenario, namely the monitoring of noise pollution levels in urban environments.

The contribution of this thesis is therefore threefold: it describes the design of novel temporal and spatial sensor selection algorithms; it provides an implementation of these algorithms on simulators and state-of-the-art wireless sensor network platforms; and it describes a thorough performance evaluation based on simulations, real-world experiments, and a specific application scenario. Taken together, these contributions constitute a step towards enabling long-term environmental monitoring applications using wireless sensor networks.

Kurzfassung

Ein drahtloses Sensornetz ist ein Verbund kleiner elektronischer Geräte – sogenannte Sensorknoten –, welche über eine Energiequelle sowie über Daterfassungs-, Kommunikations- und Rechenfähigkeiten verfügen. Ausgebracht in der Umgebung können Sensorknoten Messungen einer physikalischen Grösse, z.B. der Temperatur eines Gletschers, durchführen. Um die einzelnen Messergebnisse eines jeden Sensorknotens einzusammeln, bilden diese mittels ihres integrierten Funkmoduls ein Ad-hoc-Kommunikationsnetz, über welches dann die jeweiligen Nachrichten, möglicherweise über mehrere Zwischenstationen, zu einer bzw. zu mehreren Datensenken verschickt werden. Dank der grossen Anzahl ausgebrachter Knoten und ihrer Fähigkeit, für lange Zeit wartungsfrei operieren zu können, ermöglichen Sensornetze eine detaillierte Vermessung verschiedenster physikalischer Grössen.

Trotz aller technischer Fortschritte stellt der Betrieb eines Sensornetzes über einen längeren Zeitraum hinweg jedoch noch immer eine Herausforderung dar. Während eine detaillierte Messung und die zeitnahe Übermittlung von Daten oft Hauptanforderungen an eine Sensornetzanwendung sind, sollte gleichzeitig das Funkmodul als Hauptenergieverbraucher möglichst selten verwendet werden. Der effiziente Betrieb eines Sensornetzes setzt daher eine sorgfältige Planung der einzelnen Sensorknotenaktivitäten – d.h. dem Erheben von Messdaten sowie deren Übermittlung – voraus. In diesem Zusammenhang können sogenannte Sensor-Selektionsalgorithmen eine wichtige Rolle spielen. Indem sie in Abhängigkeit von den jeweiligen Qualitätsanforderungen der Sensornetzanwendung lediglich eine Untermenge von Sensorknoten auswählen, welche zu einem gegebenen Zeitpunkt messen bzw. kommunizieren müssen, können sie signifikante Energieeinsparungen erreichen und so unabhängig von den eingesetzten Medienzugriffs- und Routing-Protokollen die Lebensdauer des Sensornetzes merklich verlängern.

Diese Arbeit erweitert bestehende Sensor-Selektionsalgorithmen um die Fähigkeit, sich an den konkreten Datenverlauf und die lokale Topologie des Netzes anzupassen und dabei kein – bzw. nur ein Minimum

an – A-priori-Wissen über das zu beobachtende Phänomen zu nutzen. Statt dessen werden die Selektionsparameter stetig und in einem verteilten Verfahren angepasst. In einem ersten Schritt wird ein Verfahren vorgestellt, welches es Sensorknoten erlaubt, ihre Datenübertragungsrate selbständig an das beobachtete Phänomen anzupassen. Dazu übertragen die Knoten der Datensenke zusätzlich zu den aktuellen Messwerten ein Vorhersagemodell, welches die Senke ihrerseits für die Abschätzung zukünftiger Messwerte nutzen kann. Solange die Senke sukzessive Messwerte innerhalb gegebener Fehlergrenzen abschätzen kann, können Sensorknoten von einer Übertragung dieser Messwerte absehen. Das vorgestellte Verfahren ermöglicht es den Knoten, mehrere Kandidaten für das zu verwendende Modell gleichzeitig vorzuhalten und in jeder Situation das jeweils beste Modell dynamisch auszuwählen. In einem zweiten Schritt betrachtet diese Arbeit die räumliche Auswahl von Sensorknoten in Anwendungen, die einen bestimmten Teilbereich eines Gebiets mit ihren Messungen abzudecken versuchen. Dabei wird eine neue Strategie, welche die Rangordnung von Sensorknoten für eine spezifische Messaufgabe anhand von Informationen über die lokale Topologie des Netzes berechnet, präsentiert. Diese Rangordnung, kombiniert mit passenden Randomisierungsverfahren, erlaubt schliesslich die Auswahl einer Untermenge von Sensorknoten, deren Aktivierung zur Erfüllung der Anwendungsanforderungen ausreicht.

Die in dieser Arbeit vorgeschlagenen Sensor-Selektionsalgorithmen beanspruchen nur wenig Rechenleistung und Speicher, was ihren Einsatz auch auf äusserst ressourcenbeschränkten Sensorknoten ermöglicht. Der praktische Einsatz der Algorithmen wurde sowohl mit Hilfe umfangreicher Simulationen als auch durch praktische Experimente mit prototypischer Hardware und im Rahmen eines spezifischen Anwendungszenarios, der Überwachung von Umgebungslärm, untersucht.

Die vorliegende Arbeit liefert somit drei Hauptbeiträge: Sie schlägt neue Algorithmen für die zeitliche und räumliche Sensorselektion vor; sie beschreibt eine Implementierung dieser Algorithmen auf Simulatoren und auf aktuellen Sensorknoten; und sie bietet eine detaillierte Leistungsanalyse der Algorithmen basierend auf Simulationen, Experimenten und der Diskussion eines konkreten Anwendungszenarios. Zusammengenommen stellen diese Beiträge einen wichtigen Schritt zur Realisierung langlebiger Anwendungen zum Überwachen der Umwelt mittels drahtloser Sensornetze dar.

Riassunto

Una rete di sensori è un sistema costituito da minuscoli dispositivi, detti nodi sensore, che vengono tipicamente alimentati a batterie e dispongono di un processore, una o più memorie dati, sensori di vario tipo nonché un trasmettitore. Una volta distribuiti su di un area che si vuole monitorare, i nodi sensore acquisiscono campioni di variabili fisiche, come la temperatura di un ghiacciaio o la concentrazione di un agente inquinante. Per comunicare i dati raccolti ad una stazione base i nodi sensore formano una rete ad-hoc, e generalmente multi-hop, utilizzano i loro trasmettitori. Una rete di sensori può operare autonomamente per lunghi periodi e supportare un elevato numero di nodi, permettendo cosi di acquisire misure ad elevata granularità, sia spaziale che temporale, di una o più variabili fisiche.

Rendere una rete di sensori capace di far pervenire alla stazione base grandi quantità di dati durante lunghi periodi di tempo è tuttavia ancora un problema aperto. In particolare, l'utilizzo del trasmettitore comporta un elevato dispendio di energia per i nodi sensore e limitarne la frequenza di attivazione è dunque necessario per prolungare la durata operativa della rete. Allo stesso tempo, molte delle applicazioni nel contesto delle quali le reti di sensori trovano il loro utilizzo richiedono frequenti comunicazioni con la stazione base per permettere ai nodi sensore di inviare regolarmente i dati acquisiti. Una attenta pianificazione delle attivazioni dei singoli nodi è quindi necessaria al fine di operare in modo efficiente una rete di sensori. In questo contesto, i cosiddetti algoritmi di selezione dei sensori permettono di limitare il numero di comunicazioni totali con la stazione base individuando adeguati sottoinsiemi di nodi la cui attivazione permette di soddisfare i requisiti dell'applicazione. Tali algoritmi, tuttavia, presuppongono spesso la disponibilità di informazioni sulle caratteristiche dei fenomeni che si vogliono osservare.

Questa tesi propone quindi di eliminare, o ridurre, la necessità di tale conoscenza pregressa dotando gli algoritmi di selezione dei sensori della capacità di adattarsi continuamente alla dinamica del segnale misurato

e alla topologia della rete. La tesi si rivolge in un primo tempo al problema della selezione dei sensori nel dominio del tempo considerando algoritmi che permettono ai nodi sensori di stabilire autonomamente la frequenza con cui comunicano i loro dati alla stazione base. A questo scopo i nodi inviano, insieme al dato misurato, un modello di predizione che permette di stimare il valore dei prossimi campioni. La comunicazione con la stazione base può quindi essere sospesa fin quando l'errore di stima, cioè la differenza tra il valore effettivamente misurato e quello stimato dal modello, non supera una predeterminata soglia di tolleranza. Considerando che una adeguata scelta del modello di predizione è cruciale per massimizzare i risparmi in termini di comunicazione, questa tesi propone una procedura generica per effettuare la scelta di tale modello in modo adattivo e direttamente sui nodi sensori. L'algoritmo proposto mantiene un set di modelli e ne valuta continuamente le prestazioni permettendo così al nodo di selezionare, e comunicare alla stazione base, solo il modello in grado di garantire il maggior risparmio in termini di comunicazione. La tesi considera poi anche il problema della selezione dei sensori nel dominio dello spazio ed in particolare nel contesto di applicazioni che richiedono copertura spaziale di una specifica regione. La tesi introduce quindi una nuova strategia di selezione dei nodi in grado di garantire, entro predefiniti limiti di errore, la copertura della regione da monitorare. A questo scopo vengono utilizzate informazioni sulla topologia della rete ed applicate adeguate tecniche di randomizzazione.

Gli algoritmi di selezione dei sensori presentate in questa tesi richiedono una quantità di memoria dati e capacità di calcolo compatibili con le più comuni piattaforme sperimentali attualmente disponibili. Le prestazioni degli algoritmi presentati sono inoltre dimostrate da esperimenti su simulatore e piccole reti di sensori nonché dall'analisi di uno specifico scenario applicativo: il monitoraggio del rumore ambientale.

I risultati di questa tesi possono essere riassunti nella somma di tre contributi. Il primo consiste nella progettazione di nuove tecniche di selezione dei sensori. Il secondo nella implementazione di tali tecniche su simulatori e su nodi sensori. Ed il terzo nella valutazione delle prestazioni degli algoritmi tramite simulazioni ed esperimenti, nonché l'accurata analisi di uno specifico scenario applicativo. Considerati nel loro insieme, questi contributi costituiscono un passo verso la realizzazione di reti di sensori capaci di monitorare fenomeni ambientali per lunghi periodi di tempo e senza alcun intervento umano.

Contents

1. Introduction

Wireless sensor networks (WSNs) are systems of tiny, battery-powered computers endowed with sensing and wireless communication capabilities, which are commonly known as *sensor nodes* [2, 52]. Typical application scenarios for WSNs envision a large number of sensor nodes being distributed at various locations over a region of interest to capture data about some physical quantity, like temperature, atmospheric pressure, or a pollutant concentration [10, 30, 185, 186]. Sensor readings are then processed locally or reported to a central server and used to achieve application specific goals. In a typical environmental monitoring scenario, for instance, the application goal consists in capturing the temporal and spatial development of the observed physical quantity (hereinafter also referred to as *signal* or *sensor field*) within some user-defined accuracy. To report their readings to one or more data collectors, sensor nodes communicate through their integrated radio transceivers and collaboratively build an ad-hoc, possibly multi-hop relay network.

Since the wireless channel is an inherently unreliable medium, communication among nodes and, thus, data reporting, may fail or suffer high and variable latencies. Also, since the wireless channel is a shared resource, the data throughput of a single sensor node typically decreases as the density of communicating nodes increases [77]. This makes it desirable to limit communication among nodes so as to increase the overall reliability and data throughput of the network. Furthermore, radio communication is usually the major cause for energy consumption in wireless sensor networks [104, 154, 211]. Consequently, limiting radio usage is the most effective way to increase network lifetime and, indeed, saving communication is the prime concern for the design of basic services and applications in WSNs.

Following this major optimization guideline that requires limiting radio usage in WSNs, the research community proposed a variety of protocols for achieving efficient and reliable sensor data gathering. These include several power-aware medium access protocols and reliable rout-

ing schemes [31, 68, 104, 196]. Although research questions on these issues are still far from settled, WSN prototypes able to guarantee reliable data collection are now available [31, 68].

Assuming a WSN is able to reliably deliver data packets, a question that remains open is whether all nodes should actually participate in a sensing task or not. In a target detection and tracking application, for instance, one could try to select a subset of the nodes to guarantee spatial coverage and put the rest to sleep so as to save energy. Upon detection of a target, activating the sleeping nodes may then guarantee better tracking performance, although at the cost of increased overall energy consumption [205, 212]. In other scenarios, environmental scientists may be interested in observing the behavior of a sensor field over large areas [37, 190]. Letting all the available sensor nodes continuously collect data may rapidly deplete network resources. With a careful coordination of node activations, however, one can significantly improve network lifetime.

Algorithms dealing with the scheduling of the sensing activity of sensor nodes are usually referred to as *sensor selection* algorithms, since they select which nodes should participate in the sensing task [211]. Trading off energy consumption (due to sensing and communication) with data granularity, sensor selection algorithms allow the optimization of resource usage within a WSN and, consequently, the improvement of its lifetime and reliability. Sensor selection is usually performed at the application layer, since the question about which nodes should actively sense and report their observations depends on the specific application requirements. For instance, sensor selection strategies must take into account the expected lifetime of the network and the desired level of data granularity.

In the context of this work, we mainly focus on field estimation applications in which the ultimate goal of the network is the reconstruction, at a central server, of the temporal and spatial development of a specific physical phenomenon. The reconstruction of a sensor field from its scattered samples can be performed using standard signal processing techniques [56, 126, 171]. To ensure a reliable reconstruction, however, a sufficiently large number of nodes must sample the sensor field at sufficiently close time intervals, i.e., the spatial and temporal sampling rates of the network must be sufficiently high. The default values of these rates are often fixed a priori on the basis of conservative estimates of the data accuracy requirements of the application in combination with

other constraints, like the total number of available nodes and the area of the region to cover. However, the values of the sampling rates actually necessary to comply with the application quality requirements may change over time and across different sectors of the network, since they depend on the actual dynamics of the observed signal and even on the physical topology of the network. In particular, it may be possible to, at least temporarily, reduce these rates without affecting the overall data quality. Clearly, reducing the spatial sampling rate allows for energy savings since only a subset of the nodes will be active in each sampling interval. Similarly, reducing the temporal sampling rate preserves resources since the nodes are required to sense and communicate data less frequently. By determining which sensors should participate in a given sensing task, sensor selection algorithms control the spatial and temporal data sampling and reporting rates of the network. In the remainder of this work, we will refer to *spatial* and *temporal* sensor selection strategies to indicate algorithms operating on either the spatial or the temporal sampling and reporting rate of the network, respectively.

In this thesis, we describe the design, development and testing of practical temporal and spatial sensor selection strategies for distributed, wireless sensing systems. In particular, we focus on typical environmental monitoring scenarios requiring long-term data collection. Drawing and improving upon existing work, we provide practical strategies that are able to limit the overall energy consumption of the network while guaranteeing the accuracy of the reconstructed data. Our approaches provide significant reductions in data communication, even if no or only little a priori knowledge of the signals of interest is available. Furthermore, the little computational and memory resources our algorithms draw upon, makes them executable on state-of-the-art WSN prototyping platforms. To demonstrate the actual feasibility of the designed algorithms and to assess their performance, we report experimental results gathered through extensive simulation studies and a small-scale WSN deployment.

In the following section 1.1, we provide a closer view of the sensor selection problem and the challenges related to the design and implementation of algorithms seeking its solution. We then go into the details of temporal and spatial sensor selection strategies in sections 1.2 and 1.3, respectively. Finally, we summarize the contributions of this thesis in section 1.4 and provide an overview of its content in section 1.5.

1.1. The Sensor Selection Problem

The typical task of a WSN consists in gathering measurements of a sensor field over a region of interest (RoI) for a possibly long period of time. The spatial and temporal frequency of the measurements may vary significantly depending on the specific application scenario. However, irrespectively of the application, limiting radio usage is crucial in preserving the scarce energy resources of sensor nodes. This requires us to not only constrain the number of data packets exchanged among nodes, but also to reduce the amount of time during which the radio circuitry is powered on. Indeed, keeping the radio in idle listening mode may be nearly as expensive, in terms of energy expenditures, as using it for sending and receiving data [154]. Additionally, some applications may require the use of energy-hungry sensors, whose activation can drain a non-negligible amount of current. In general, sensor nodes frequently participating in sensing and communication activities may quickly deplete their energy supply and thus become unavailable. This progressive failing of single sensor nodes may, in turn, rapidly compromise the ability of the network to comply with the application requirements and, in fact, make it unusable. These considerations show that a careful scheduling of the participation of sensor nodes in sensing and communication is instrumental in ensuring reliable and long-lasting operations of a WSN.

The problem of selecting, at the desired time instants, a subset of sensors able to comply with the quality requirements of the application while limiting the overall network resource consumption, is known in the literature as the *sensor selection* or *sensor tasking* problem [211] and can be seen as a particular instance of the sensor role assignment problem [61]. In this context, *sensor selection algorithms* or *strategies*, are procedures able to select, according to some trade-off, the above mentioned subsets. In the following sections 1.2 and 1.3, we detail, for specific application scenarios, how sensor selection algorithms can operate to optimize the temporal and spatial activation intervals of the nodes in a WSN. Before going into further details, however, we would like to point out here the major challenges and design guidelines related to the definition of sensor selection strategies.

First, sensor selection can be performed using centralized or distributed approaches. The work presented in [46], for instance, proposes a centralized approach to determine which nodes hold data whose in-

formation content is likely to maximize the probability of a user query to be answered correctly. This approach requires comprehensive information about the network (e.g., physical and logical topology), or an estimation thereof, to be known at a central server. If the properties of the network change over time, as it is actually common in WSN settings, the information at the server may rapidly become stale and thus potentially hamper the performance of sensor selection algorithms that rely on this information. On the other hand, providing the server with frequent updates on network status may cause a high, and clearly undesirable, communication overhead. Approaches performing sensor selection based on centralized decisions [46, 69, 120, 203] may thus fail to perform efficiently in WSNs settings, especially as the number of nodes within the network and its physical area increase. On the other hand, distributed approaches typically offer only approximate or probabilistic solutions to the sensor selection problem [14, 152, 205]. However, they usually also provide for significantly lower overhead and higher adaptivity to changing network and environmental conditions. In other words, *"to achieve scalability and autonomy, sensor tasking and control have to be carried out in a distributed fashion, largely using only local information available to each sensor"* [211, page 135]. In our approach to sensor selection in WSNs, we sought distributed algorithms that let individual nodes autonomously decide wether or not to participate in the sensing task. In particular, we show that it is possible to achieve significant savings in terms of energy spent for data collection and communication by making this decision depend on the actual dynamics of the observed signals and, possibly, the local topology of the network [116, 163, 166].

Data-dependency is indeed a further factor influencing the design and performance of sensor selection strategies. To illustrate this point we report an example, described in [211, Section 5.3], concerning the localization of a stationary source using a set of sensor nodes deployed in a square region. One of the nodes, elected as the leader, collects sensor measurements and selectively interrogates its peers for their data to bring the uncertainty on the estimate of the position of the source under a desired threshold. To this end, it first applies a nearest neighbor (NN) strategy, that makes it trigger data reporting from the nearest node whose measurements have not yet been included. This simple selection strategy takes into account the physical topology of the network but does not adapt to the actual information content of the collected data. An alternative strategy to solve this source localization problem

consists of performing sensor selection using a more elaborate metric. For instance, relying on the Mahalanobis distance [211, page 150] allows the algorithm to maximize the information gain obtainable by the incorporation of a new data sample. Using this metric, the choice of the nth sensor required to deliver its reading depends on both its position and the actual measurements delivered by the previously interrogated $n-1$ nodes. In this context, the sensor selection process becomes data-dependent. In this thesis, we show that the ability to adapt to the actual signal dynamics may be instrumental in improving the performance of sensor selection algorithms working in the field estimation scenarios we are interested in.

The availability of reliable models to represent the physical phenomena the network is set up to observe may also open several possibilities for the design and implementation of sensor selection algorithms [46, 76, 144]. In particular, exploiting the a priori information made available by the models may make the selection process more efficient. For instance, knowing that the phenomenon of interest can be well represented in a certain function space enables the adoption of techniques seeking for the computation of the coefficients of the signal in this space instead of a complete data collection [76]. In many practical situations, however, there may be little or no a priori information on the signals of interests [62]. In these scenarios, model-based approaches loose their appeal, since their performance may degrade significantly as the actual characteristics of the signal differ from those of the assumed model. In the context of our work, we are mainly interested in environmental monitoring scenarios that are exploratory in nature and for which usually limited or no a priori information is available [62]. Therefore, we avoid the use of specific models to represent the signals of interest.

Another factor that may influence the sensor selection process is the interplay with protocols working at different levels of abstraction [151, 183, 211]. In particular, routing protocols rely on a subset of nodes to be active to relay sensor data to a central server. Using the same subset of nodes to gather sensor measurements may provide for energy savings since it would not require the activation of additional sensors. On the other hand, preserving nodes involved in sensing activities to be used also as data routers may help in balancing the energy consumption across nodes. In the context of spatial sensor selection, we explicitly investigate the potential synergies between sensor selection and routing

as a means to increase network lifetime.

Some authors also investigated the possibility of concurrently optimizing both the temporal and spatial sampling rate of a WSN and thus developed spatio-temporal sensor selection strategies [46,175]. Indeed, we will show that our temporal sensor selection strategy, presented in detail in chapter 3, implicity performs also a spatial sensor selection. Furthermore, our temporal sensor selection strategy can be easily extended to exploit spatial dependencies among the measurements and thus provide for further communication savings. Our approach to the spatial sensor selection problem, whose details are reported in chapter 4, assumes the network is required to report data at regular time intervals and thus will not focus on the issue of determining the temporal sampling rate of the network.

In the following two sections, we provide more specific considerations in order to clarify our approach to the sensor selection problem in both the temporal and spatial domains.

1.2. Temporal Sensor Selection

In typical WSN deployments, sensor nodes are required to sample a certain physical quantity at regular time intervals and report these readings to a central data collector [19, 30, 71, 121, 139, 185]. In a fire detection scenario, for instance, monitoring the risk of fire breakouts may require air temperature readings to be periodically collected at several locations over a remote forest region. Similarly, the availability of fine-grained, real-time data about the water temperature below the sea surface may help fishermen in improving the efficiency of their fishing activities [199].

In these scenarios, sensor nodes assumed to collect sensor readings using a common and fixed sampling interval Δ_T, and required to report them to the sink directly upon collection. The network can thus persist in *sleep* state[1], wake up only every Δ_T seconds to provide for sampling and data reporting and immediately go back to sleep. This duty-cycled operation mode, to which we also refer to as the *default monitoring scheme*, clearly allows to save energy, since the network is not required to be active continuously. However, letting all nodes report their readings at each sampling round may still cause a rapid

[1] In the common WSN jargon, a network is in *sleep* state when its nodes have their circuitry powered off and can thus neither communicate nor perform sensing or computation.

depletion of their batteries.

Since subsequent readings collected by the same sensor are likely to be correlated, it may be possible to reduce the data reporting rate without affecting the overall accuracy of the collected data. In this context, temporal sensor selection strategies may help in determining the actually necessary reporting rate. In particular, a widely investigated approach to reduce communication in such continuous monitoring scenarios is based on time series forecasting [46, 97, 114, 116, 122, 134, 144, 166, 194, 207]. These temporal sensor selection strategies still require the nodes to collect data every Δ_T, but allow to reduce the actual number of samples that must reach the central server in order to guarantee the collected data to lie within a given error threshold. To this end, a sensor node can locally compute an adequate prediction model that can provide for reliable estimates of future sensor readings. This model, along with the corresponding parameters, can then be reported to the sink, which will use it in the successive sampling rounds to compute estimates of the data collected at the node. Since the node can locally monitor the error between the estimates and the actual sensor readings, it can suppress communication with the sink as long as this error does not exceed the pre-specified threshold. In the absence of notifications from the node, the sink can thus assume that the estimate computed using the shared prediction model is within the allowed error bound.

This temporal sensor selection strategy, named *dual prediction scheme* or *DPS* [166], may provide for significant communication and energy savings if adequate prediction models are used [46, 97, 114, 144, 166, 194, 207]. However, many prediction techniques, like, e.g., Kalman filtering [97, 101], rely on the specification of parameters whose computation may be computationally expensive or require a priori knowledge of the signals of interest. On the other hand, using a pre-computed set of parameters may make the model unable to follow the actual signal dynamics and thus seriously hamper the achievable communication savings. Thus, adequate procedures to select suitable prediction models and compute their parameters on sensor nodes are required. To this end, we introduce a generic and lightweight adaptive model selection framework, that allows sensor nodes to autonomously determine a good model choice. The rationale of our approach consists in letting sensor nodes run a proper set of *candidate* prediction models and assess their performance in an online fashion, as sensor data is collected. Using adequate performance metrics, our framework makes sensor nodes able

to select, among the set of candidates, the model that allows for the highest achievable communication savings [116].

Other approaches to the temporal sensor selection problem avoid the use of prediction models. In event detection applications, for instance, very basic sensor selection mechanisms increase the sampling frequency of the nodes when a specific pattern is detected or decrease it when nothing of interest is happening [39]. In our thesis, however, we mainly focus on prediction-based temporal sensor selection algorithms leveraging the DPS data collection strategy sketched above.

1.3. Spatial Sensor Selection

In the previous section, we discussed how temporal sensor selection strategies may allow to reduce the data reporting rate of sensor nodes by exploiting the correlation between subsequent sensor readings. When the number and density of the nodes is sufficiently high, correlation between readings collected by nearby nodes is also likely to appear. The presence of this correlation may thus again enable the use of adequate sensor selection techniques to reduce the average number of nodes required to sample or report data. In general, spatial sensor selection is applicable whenever the actual density of the deployed nodes is higher than strictly necessary to comply with the accuracy requirements of the application. In these scenarios, the activation of all nodes at each sampling round may no longer be necessary nor desired. Thus, spatial sensor selection algorithms can pick up an adequate subset of nodes whose activation can provide for the application requirements to be fulfilled.

For instance, in surveillance and tracking applications the network is usually required to continuously provide for complete coverage of the region of interest. To this end, the application typically assumes each node to be able to cover the area span by a discus centered at the node itself and having *sensing radius* (or *range*) R_s. Thus, guaranteeing coverage of the area of interest requires each point of the area to lie within the sensing range of at least one node. Clearly, if the average distance between nodes is sufficiently small, it is possible to provide coverage activating only a subset of the available nodes. Spatial sensor selection protocol working along this rationale are known as *coverage preserving protocols* [188, 205, 209, 210].

The selection criteria used to determine the set of active nodes clearly

depend on the application and its specific quality requirements. In the context of our work on spatial sensor selection, we focus on applications having the reconstruction of a sensor field as their ultimate goal. In this specific setting, the network is required to provide, at each sampling round, a sufficiently high number of samples of the sensor field. In particular, these samples must be sufficiently representative to allow adequate reconstruction algorithms to compute, within a given accuracy, the values of the field over the whole region of interest. To this end, the necessary number of readings, and thus of selected sensors, and their spatial distribution over the region of interest depend on the requirements of the specific reconstruction algorithm used at the central server. In chapter 4 we show that, under specific assumptions, the problem of selecting an adequate subset of sampling nodes for the purpose of field reconstruction can be reduced to a coverage problem. Thus, assuming that the value of the sensing range R_s of the nodes can be linked to the desired reconstruction accuracy, coverage preserving algorithms can be leveraged as spatial sensor selection strategies also in the context of sensor field reconstruction applications.

Spatial sensor selection may also come into play to perform distributed computation of aggregates [14, 44, 109, 120] or to provide for multi-resolution storage within a WSN [62]. In the context of our work, however, we mainly focus on the sensor field reconstruction scenarios discussed above.

1.4. Summary of Contributions

The main goal of this thesis is to design and implement practical strategies for performing temporal and spatial sensor selection in WSNs. In designing our sensor selection algorithms, we focus on three main optimization goals. First, we aim at limiting the number of total data acquisitions and transmissions necessary to comply with the accuracy requirements of the application. Second, we attempt to devise computationally efficient strategies with low memory footprints, which can be implemented on resource-poor sensor nodes. Third, we seek to avoid or limit the use of a priori information about the signals of interest, which helps to lower the number of parameters that the user or the application is required to define prior to operations.

These three goals are supported by three distinct contributions.

First, we investigate the sensor selection problem in the *time* domain

and propose novel techniques to perform prediction-based data collection. We provide an implementation of the dual prediction scheme based on the least-mean-square linear filter [166] and discuss the downside of using an approach based on the a priori selection of a specific prediction model. To deal with this issue, we propose an adaptive model selection scheme (called AMS) that allows sensor nodes to autonomously select the statistically most suitable model among a set of candidates [116]. We propose an implementation of the AMS based on autoregressive models (AR), named AR-AMS, and test its performance on 14 sensor time series retrieved from real WSN deployments. Our simulation results demonstrate the versatility of the proposed framework and its ability to achieve higher communication savings achievable with respect to a "classical" dual prediction scheme [116]. Further, we propose an alternate, more generic and lightweight implementation of the AMS based on exponential smoothing (ES) models, to which we refer to as the ES-AMS. This second version of the AMS achieves comparable performance, in terms of communication savings, with respect to the previous implementation, but offers both lower computational overhead and memory overhead. To demonstrate the suitability of the ES-AMS to be executed on real WSN platforms, we also implemented it as a TinyOS[2] application and tested its behavior on a small-scale WSN deployment.

Second, we address the sensor selection problem in the *spatial* domain and provide an overview of approaches that address it from several different perspectives. In particular, we focus our attention on applications having the reconstruction of a sensor field as their ultimate goal. For these applications, we show that the spatial sensor selection problem can be reduced to a coverage problem. We thus leverage the CCP coverage preserving protocol [205] as a sensor selection strategy and introduce a novel sensor ranking heuristic that enables a reduction of its communication overhead. The heuristic ranks the relevance of a node for the sensing task based on its position with respect to nearby located nodes. This strategy allows the protocol to quickly select a subset of nodes that can provide complete coverage of the region of interest, and thus improve upon the performance of the original implementation of the CCP protocol. Further, we leverage the same heuristic to design a novel, adaptive random sensor selection strategy (ARS) [163]. Our analysis of the performance of the ARS shows that it can offer high

[2] TinyOS is the de-facto standard operating systems for WSNs [118,189].

levels of coverage of the region of interest while activating a significantly lower number of nodes with respect to a simple random selection strategy. Finally, we investigate the potential synergies between sensor selection and routing, and briefly discuss the interplay of our ARS strategy with a state-of-the-art routing protocol.

Third, we provide a thorough analysis of an intriguing application scenario for WSNs: the monitoring of noise pollution in urban areas. In particular, we distill the application requirements and analyze the suitability of our sensor selection strategies in this context. Furthermore, we report our experiences in using different WSN platforms, as well as mobile phones, as noise pollution sensors [57, 164, 165, 167].

1.5. Outline

The remainder of this thesis is organized as follows. Chapter 2 provides a brief overview of WSNs and the main software and hardware platforms we make us of in the context of this work. Further, it allows us to introduce some definitions, notation and assumptions we will refer to in the rest of the thesis. In chapter 3, we report in detail our approach to the temporal sensor selection problem based on time series prediction. Chapter 4 presents our spatial sensor selection strategy and the related experimental evaluation, as well as a thorough analysis of related work. Chapter 5 reports our experience in using WSNs for environmental noise monitoring, while the following chapter 6 provides details of the software tools and libraries we implemented in the context of this work. Finally, we provide our conclusions, along with some considerations about promising directions for further investigations, in chapter 7.

2. Background

In the previous chapter, we described the main challenges related to the solution of the sensor selection problem in wireless sensor networks (WSNs) and outlined the main contributions of this thesis. For the reader unfamiliar with the field of WSNs, we provide here a brief overview of the challenges related to the design, implementation, and deployment of these systems. Further, we describe the set of hardware and software platforms we make use of in the context of our work. Finally, we describe the mathematical notation, the models and assumptions we rely on for designing and evaluating our sensor selection protocols. In the remainder of this thesis, we will briefly recall or refer to the content reported in this chapter whenever necessary or appropriate. The reader familiar with the broader field of WSNs can therefore skip this chapter and still smoothly follow the rest of the thesis.

2.1. Wireless Sensor Networks

In 1999, Kahn, Katz, and Pister, describe the rising of a new category of systems made of "*very compact, autonomous and mobile nodes, each containing one or more sensors, computation and communication capabilities, and a power supply*" [100]. Likely to become as cheap and small as grains of sand, Kahn et al. envision these nodes, also dubbed *motes* or *sensor nodes*, to be deployed in hundreds or thousands of units over large regions and to operate unattended for days, weeks or years. Building the particles of a *smart dust*, they note, sensor nodes can perform fine-grained measurements of physical properties of the environment, like temperature, humidity, or sound. The collected sensor data can then be stored and processed locally and/or reported back to a central server for further analysis. To this end, nodes use their in-built communication module to cooperatively build a multi-hop data-relaying network. The possible applications of these wireless networks of sensor nodes seemed countless and ranged "*from sensor-rich smart spaces to self-identification and history tracking for virtually any kind*

of physical object" [100].

In their seminal paper, Kahn et al. noted that the research challenges towards the realization of working WSNs were, as partially still are, numerous, new and all but trivial. In particular, in contrast to traditional wired and wireless networks, WSNs must be able to operate without relying on a fixed infrastructure, like ad-hoc networks. With respect to the latter, however, WSNs also present several unique characteristics [52, 125]. For instance, sensor nodes have only little computational and memory resources and dispose of limited and usually non-renewable power supply. Since running out of power would make a node, and eventually the whole network, unusable, optimizing power consumption is a primary concern in WSNs. Additionally, communication in WSNs is usually short-range and the topology of the network may change frequently due to hardware and software failures or instability of the communication channel. Dealing with these uncertainties and dynamics is mandatory for WSNs to work reliably over long periods of time. Further, since the density of nodes in a WSN may be much higher than in traditional ad hoc networks, scalability becomes a crucial issue [52, 125]. At the same time, the application may exploit this high density to optimize nodes' activations. As further outlined in [125], another peculiar characteristic of WSNs is that they typically do not rely on point-to-point communication. Indeed, due to high numbers of nodes and the possibly large deployment regions, nodes are typically unaware of the whole size and topology of the network. Therefore, addressing single nodes in the network may be impractical and inefficient. Instead, nodes may be addressed using specific attributes, like their positions or available sensors and data.

The design and implementation of protocols and algorithms for WSNs must thus take into account the limited power supply and poor computational and memory resources of sensor nodes, the high number of nodes and their potential unreliability as well as the unpredictability of the environment. These factors pose strong requirements on the robustness and scalability of protocols working at the physical, MAC or routing layer. Furthermore, the design and implementation of basic services like localization and synchronization as well as application-level algorithms cannot abstract from the above mentioned issues, since this would likely result in inefficient solutions.

In the last decade, WSNs have been a field of active research and by now a large literature on protocols and algorithms for WSNs is avail-

able [104, 179, 211]. Furthermore, moving from and beyond the smart-dust vision, several experiments showed the potential of WSNs to be used in a plethora of different application contexts. For instance, in environmental monitoring, to observe birds' habitats and habits, [73,185] or to investigate the growth model of redwood trees [30]. Or in precision agriculture, to study the influence of environmental parameters on food quality [10, 71]. Further, in fire detection, avalanche prevention, and countless additional civil and military application scenarios [11, 12, 22, 99, 103, 106, 129, 176, 186, 201].

However, enabling a wireless sensor network to reliably report large quantities of data over long periods of time is still a challenging goal [12,20]. For instance, comprehensive tools to inspect and debug the network at run-time are still scarce and require field expertise to be used. Furthermore, many of the existing protocols and algorithms may only be able to work reliable in specific application scenarios, and robust, generic solutions are barely available. In this context, the definition of a distinctive approach to the design and deployment of WSNs "*still requires further research and experience*" [20].

2.2. Hardware and Software Platforms

The previously cited work by Kahn et al. [100] as well as a series of other seminal papers [52, 85], represent early results in the research field of WSNs. In the years to follow, several researchers and projects contributed to the development of hardware and software platforms for WSN prototyping. Among these, we selected for our work a set of well-known platforms like the *Tmote Sky* sensor node, the *TinyOS* operating system, and the *Castalia* simulator. Furthermore, we made extensive use of the *Matlab* computing environment. In the following, we provide a brief description of these platforms and motivate our choices.

2.2.1. The Tmote Sky Sensor Node

The smart dust prototype of 1999 showed the possibility of building sensor nodes the size of just a few cubic millimeters, but several, more powerful, generations of sensor nodes followed. There exist a number of surveys on hardware platforms for prototyping WSN systems. In particular, we refer the interested reader to Tatiana Bokareva's *Mini*

Hardware Survey[1], to *The Sensor Network Museum* project [2], or the *Embedded Wisents Platform Survey* [13]. Here, we would like to introduce in more detail the Tmote Sky sensor node, which we used as a reference prototyping platform for the considerations and experiments presented in the following chapters.

Evolved from the *TelosB* family of motes [43, 154], the Tmote Sky has been commercialized by Moteiv Corp. [136], a UC-Berkeley spin-off, from 2004 until the end of 2007. The Tmote Sky features a Texas Instruments $MSP430\ F1611$, 16-bit RISC processor that can operate at extremely low power levels.[3] The internal oscillator of the $MSP430\ F1611$ can operate at a maximum of 8MHz and can be activated in as few as $6\mu s$, thus allowing the processor to switch very efficiently from sleep to active mode. The Tmote Sky is further equipped with 10kB of RAM, while a 48kB flash memory is available for safely storing programs and data. For communication, the Tmote Sky relies on the Chipcon[4] wireless transceiver $CC2420$, an IEEE 802.15.4 compliant radio operating at 2.4GHz and offering data rates of 250kbps. Two 1.5V, AA batteries provide the power supply when the node is not plugged-in through its embedded USB-adapter. Two photodiodes, the Hamamatsu $S1087$ and $S1087\text{-}01$, allow the Tmote to measure the photosynthetically active radiation (PAR) and total solar radiation (TSR), respectively. Further, the Sensirion $SHT11$ (or $SHT15$) sensor provides for (calibrated) humidity and temperature data. As basic actuators, the Tmote Sky also features three external LEDs of different colors. Additional sensors and actuators (a microphone, an accelerometer and a speaker) are available on a twin version of the Tmote Sky, dubbed *Tmote Invent*, which also features a practical and elegant packaging. Both platforms are shown in figure 2.1.

To substantiate the often-cited argument that radio communication is the main factor of power consumption on a sensor node, we report in table 2.1 some relevant figures regarding the current drained by the Tmote in different operating modalities. The values reported in the table are taken from [154] and the Tmote Sky's datasheet [137].

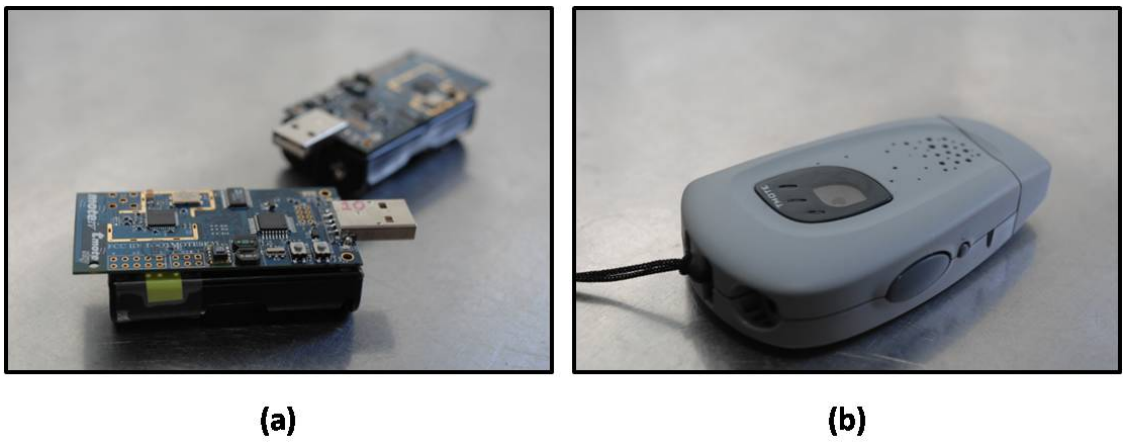
If we assume the batteries of a mote offer a total capacity of 2000 mAh, keeping the microcontroller uninterruptedly active will make the

[1] `www.cse.unsw.edu.au/~sensar/hardware/hardware_survey.html`

[2] `www.snm.ethz.ch`

[3] `ti.com/msp430`

[4] `www.chipcon.com`



(a) (b)

Figure 2.1.: The Tmote Sky (a), and Tmote invent (b) WSN platforms.

Table 2.1.: Current consumption of the Tmote Sky sensor node under typical operating conditions (nominal values).

Operating condition	**Current consumption**
Mote standby	5.1 μA
MCU idle, oscillator on	54.4 μA
MCU active	1.8 mA
Radio transmitting at 0 dBm	17.4 mA
Radio receiving	19.7 mA
Radio on, oscillator on	365 μA
Idle mode, oscillator off	20 μA
Flash memory (active current, read)	4 mA
Flash memory (active current, write/erase)	20 mA

node run out of power in about 46 days. If, concurrently, the node continuously transmits data, the batteries will be empty after just 4 days. Although simplistic, this computation gives a feeling of the burden radio communication constitutes on the energy budget of a node. We should also notice that, in terms of drained current, writing data on the flash memory can be as expensive as radio communication. Furthermore, power consumption due to sensing is not always negligible. Indeed, while the Sensirion $SHT11$ light sensor requires only 25μA to perform one 12-bit measurement per second[5], the on-board microphone of the Tmote invent drains as much as 2.3mA of current [137]. If the microphone remains active for a prolonged time, for instance to record the call of a bird or estimate the current noise level, its contribution is not negligible in the total energy budget of a node.

[5] www.sensirion.ch/en/01_humidity_sensors/02_humidity_sensor_sht11.htm

2.2.2. The TinyOS Embedded Operating System

There exist several operating systems specifically designed for WSNs. Some examples include TinyOS [118, 189], Contiki [50], BTnut[6], and Mantis [21]. As was the case for hardware platforms, however, it is beyond of the scope of this chapter to provide an exhaustive survey of such systems. Instead, we would like to focus on TinyOS, which is largely considered the de-facto standard operating system for WSNs. More details about alternative systems, like those cited above, are available in [13, 21, 50, 174].

TinyOS is an open-source, flexible, and energy-aware operating system specifically designed to support WSNs applications. It features a component-based architecture that allows programmers to combine small modules of code, called *components*, into more complex programs. In the TinyOS jargon, the process of combining independent modules through their interfaces is called *wiring*. The final binary image of the program includes only those components necessary to implement the application logic that has been, explicitly or implicitly, wired. In this sense, TinyOS enables developers to build an application-specific operating system, saving precious memory resources on the nodes. Indeed, the whole TinyOS core fits in less than 400 bytes and a typical complete application performing sensing and communication is usually only about 15kB in size [118].

Typical events in WSN scenarios, like reception of a radio packet or the collection of a specific sensor value, may occur unpredictably and require a timely reaction of the operating system. To this end, TinyOS supports an event-based concurrency model. It allows the programmer to define *events* to which the operating system reacts immediately leveraging the so-called *split-phase* operation mode. In this way, event handlers can preempt less time-critical code, which is accordingly included in TinyOS *tasks*.

TinyOS is written in nesC [67], a programming language specifically designed to support a component-based architecture, event-based concurrency model, and split-phase operation mode. nesC is an extension of the C language [105] and is also the language of choice for writing TinyOS applications.

The first version of TinyOS, known as TinyOS 1.x and released in 2002, was replaced in November 2006 by the first stable release of

[6] `www.btnode.ethz.ch`

TinyOS 2.0. This second version is currently used by more than 500 research groups and its wide developer and user base makes TinyOS the de-facto standard operating system for developing WSN applications [189].

2.2.3. The Matlab and Castalia Simulators

As we mentioned several times, WSNs are envisioned to be made of hundreds or thousands of tiny sensor nodes, possibly deployed in harsh environments. Unexpected behaviors and errors occurring after deployment may hamper network performance or even make it unusable [20]. A thorough analysis and testing of the algorithms running on the nodes prior to deployment is therefore crucial to limit the occurrence of such unexpected problems. To this end, the use of small scale test deployments and simulation is mandatory [12].

Some authors criticize the use of simulation as an investigation tool as too simplistic and unable to capture the complex conditions in which WSNs are envisioned to operate. In particular, the common assumption of perfectly circular radio coverage is doomed to failure in real WSN settings [107]. On the other hand, this assumption is widely used in the literature since it allows us to derive useful general results [161]. Since our main interest is in developing algorithms working at the application level, we abstract several low-level issues and also resort to the usual assumption of perfectly circular radio coverage. Whenever necessary or opportune, we discuss the effects of non-ideality on the behavior of our algorithms and propose possible countermeasures. With this premises in mind, we now introduce the simulators we considered in our investigations, namely the Matlab computing environment and the Castalia sensor network simulator.

Matlab[7] is a well-known platform for numerical computation. It allows the user to easily manage and visualize data and provides a large number of built-in mathematical functions and specialized computational tools. For instance, it offers several interpolation methods for signal reconstruction, or a toolbox with specialized time series analysis functionalities. As we will show in the next chapters, we implemented our sensor selection algorithms as Matlab applications and performed most of our experimental evaluations using these implementations. Although Matlab does not provide any realistic radio or channel models,

[7] www.mathworks.com

it allowed us to gain insights on the ideal performance of our algorithms and thus to set a sort of benchmark towards which the effect of non-ideality can be investigated.

To perform our preliminary study on the interplay between spatial sensor selection and routing reported in section 4.8, we used the Castalia[8] WSN simulator [26, 153]. Castalia is a simulator for WSNs implemented on top of the OMNeT++ platform[9]. OMNeT++ is a discrete event simulation environment that thanks to its excellent modularity is particularly suited to support frameworks for specialized research fields. For instance, the *Mobility Framework* (MF) supports simulation of mobile networks, while the *INET* framework enables the modeling of several Internet protocols. OMNeT++ is written in C++, is well-documented, and features a graphical interface that eases development and debugging. Additionally, a wide community of researchers and developers support OMNeT++ and continuously provide new modules and improvements of existing code. The comfortable initial training, the modularity, and the possibility of programming in an object-oriented language (C++) are among the reasons that led us to prefer the OMNeT++ platform, and thus Castalia, over other available simulators like the well-known ns-2[10] and the related extensions for wireless sensor networks (e.g., SensorSim [148]). Nonetheless, in the last years Castalia has been steadily improved [26, 153] and its enhanced 2.0 version has been recently released [11]. Furthermore, an increasing number of researchers resort to this simulator to support their investigations [17, 27, 192].

In the context of our work, we refer to version 1.3 of Castalia, which builds upon version 3.3 of OMNet++. In this version, Castalia features advanced channel and radio models, a MAC protocol with a large number of tunable parameters and a highly flexible model for simulating physical processes. In particular, Castalia provides bundled support for the *CC*2420 radio controller, which is the on-board transceiver of the *Tmote Sky*, our reference sensor node platform. Being a simulator originally developed for testing MAC protocols, Castalia still offers only basic support for routing protocols. However, thanks to its excellent modularity, inherited from OMNet++, Castalia can be easily extended

[8] `castalia.npc.nicta.com.au`

[9] `www.omnetpp.org`

[10] `nsnam.isi.edu/nsnam`

[11] Castalia 2.0 has been released on May 8th, 2009.

and adapted to include new or improved components. In particular, to perform the experiments presented in section 4.8, we implemented the CTP routing protocol [58, 68] as a flexible Castalia module.

2.3. Definitions, Notation, and Assumptions

In this section, we provide the definition of some basic notions, the mathematical notation, and an introduction to the main assumptions we will refer to throughout this work.

2.3.1. Network and Communication Model

We represent a WSN as a directed graph $G = (V, E)$, where V is the set of all nodes and E is the set of edges between nodes. The cardinality of V represents the total number of nodes N_{tot} in the network, i.e., $N_{tot} = |V|$. Without any loss of generality, we assume all nodes in V to be assigned an unique identifier, and we refer to node n_i (or node i) as the node that has identifier i in V. If node i can communicate directly with node j, a correspondent edge e_{ij} exists in E. In particular, we assume the transmission range of all nodes within the network to be isotropic and equal to R_{tx}. Under this assumption, the set E is defined as:

$$E = \bigcup_{i=1}^{N_{nodes}} \{e_{ij} | i, j \in V, d_{ij} \leq R_{tx}\} \tag{2.1}$$

where d_{ij} is the Euclidean distance between nodes i and j.

We refer to the *communication neighborhood* of a node n_i as the set of nodes V_i defined as:

$$V_i = \bigcup_{j=1, j\neq i}^{N_{nodes}} \{n_j | i, j \in V, d_{ij} \leq R_{tx}\} \tag{2.2}$$

2.3.2. Region of Interest (RoI)

We consider settings in which a WSN is deployed either on a segment of length L_x or on a rectangular region of sides L_x and L_y. We refer to these two deployment types as the 1- and 2-dimensional case, respectively. When the values of L_x and L_y coincide, we may refer to both of them using the symbol L.

2.3.3. Sampling Rates and Sensing Model

We assume the network is set up to gather the samples of a physical variable, like temperature or humidity, over a given period of time and specific spatial region. In particular, we assume that the physical variable, also referred to as *sensor field* or *signal*, can be represented as a continuous function of time t and location s $f(t, s)$. The network can thus gather discrete samples of f at arbitrary time instants t_k and locations s_i, provided a sensor node is present at position s_i and performs sensing at time t_k. In the context of this work, we assume the nodes to become active, and thus possibly perform sampling, at regular time intervals $t_k = k\Delta_t$, where Δ_t is the desired *temporal sampling rate*, or *temporal resolution*, typically expressed in seconds. For our considerations, we usually abstract from the specific value of Δ_t and refer to the time instant $k\Delta_t$ as t_k or k. Accordingly, we may indicate the discrete samples $f(k, s_i)$ as $f_{k,i}$. When referring to the values of f collected at a specific location but different time instants, we may use the simpler notation f_k for $f_{k,i}$. Similarly, if samples are collected at a specific time instant k, but at different locations s_i, we may use the notation f_i for $f_{k,i}$. We refer to a single sampling operation as a *sampling round*. During a single round, all the nodes or a subset of them may actively sample the sensor field.

If the nodes are deployed over a 1-dimensional RoI, we may indicate the position s_i of a node n_i as x_i. Accordingly, in the 2-dimensional case we have $s_i = (x_i, y_i)$. Definitions and notation relative to the *spatial sampling rate* or *spatial resolution* Δ_s of the network are reported in detail in section 4.3, and we thus omit them here. However, we anticipate that we define the *sensing area* of a node as a discus $D_{R_s}(c)$ having the node itself as its center c and a radius given by the *sensing range* R_s. The latter may represent a physical range[12] or a "virtual" distance the node may be able to cover.

In the following chapters, we do not specifically address issues related to the presence of noise in the data. However, we assume the samples $f_{k,i}$ of the sensor field f to be affected by a zero-mean Gaussian noise of known standard deviation. A sample $f_{k,i}$ can thus be represented as $f(k, s_i) = \widetilde{f}(k, s_i) + \nu(k, s_i)$, where $\widetilde{f}(k, s_i)$ are is the correspondent sample of the "ideal", noise-free sensor field $\widetilde{f}$ and $\nu(k, s_i)$ the realiza-

[12] E.g., for a infrared sensor, the maximal distance at which the sensor can detect the presence of a person within a given accuracy.

tion of a Gaussian random variable ν with mean $\mu = 0$ and standard deviation σ_ν, computed at time k at node i.

2.3.4. Synchronization and Localization

In order to make the network wake up at predefined time instants t_k, the nodes are required to be, at least loosely, synchronized. To this end, we assume one of the protocols known in literature to be applicable [159]. As for localization, we assume the node can retrieve their position autonomously, for instance using a GPS sensor, or through one of the available localization algorithms [29, 113, 149].

2.3.5. Routing

In our investigations, we assume the network to rely on a suitable routing protocol to report sensor readings to one or more data sinks [104, chapter 11]. In general, we do not refer to nor depend upon a specific protocol choice. However, in section 4.8 we provide a discussion on the interplay between sensor selection and routing, and use the CTP data collection protocol as a reference routing scheme. For the interested reader, we provide a description of the CTP protocol in appendix B.

In the context of this work, the terms, *sink*, *central server*, *data collector*, and *base station* are perfectly interchangeable.

3. Temporal Sensor Selection

In this chapter, we investigate the sensor selection problem in the time domain and present the first contributions of this thesis. In particular, we introduce two novel temporal sensor selection algorithms that leverage what we call the *dual prediction scheme* (DPS)[1] [116, 166]. As discussed in section 1.2, the DPS is a generic technique to perform temporal sensor selection in wireless sensor networks (WSNs) and is applicable in scenarios in which data collection must be performed within a pre-specified accuracy [166]. In particular, using the DPS it is possible to guarantee that the deviation between the sensor readings available at the sink and the actual values collected at the nodes does not exceed a given error threshold. To this end, the DPS instantiates identical *prediction models* at the sensor nodes and the data sink. Using this shared model, a node and the sink can compute the same estimations of future sensor readings. Sensor nodes can then continuously monitor the actual prediction error, i.e., the deviation between the estimated readings and the locally collected samples. If the prediction error does not exceed the given threshold, data communication between the node and the sink can be suppressed, since the estimation computed at the sink does comply with the accuracy requirements of the application. On the other hand, if the prediction error does exceed the threshold, the node must accordingly notify the sink and possibly update the prediction model. The use of the DPS can significantly improve the lifetime of a WSN, since reducing communication is an effective way to preserve energy resources on sensor nodes.

The effectiveness of the DPS greatly depends on the choice of prediction model. One such model is the least-mean-square (LMS) adaptive filter [83], which demonstrates well the energy preservation potential of the DPS approach. In a first contribution, we evaluated the use of LMS on real data sets and showed that the LMS can provide for more than 90% of communication savings with respect to the *default* data

[1] Xu et al. [207] introduced the notion of *dual prediction-based reporting* in the context of WSNs. However, we have been the first to refer to this general approach as the *dual prediction scheme* [166].

collection scheme, in which all the collected samples are transmitted to the sink [166]. However, the actual achievable communication savings in turn depend on the specific values chosen for the LMS filter parameters. Since even for the same time series the optimal parameter choice may vary over time, fixing these values a priori is usually impractical [116, 166]. Thus, adequate *adaptive parameter estimation procedures* are needed. To address this issue, which is common also to other instantiations of the DPS, we introduce our second contribution, the *adaptive model selection* (AMS) scheme.

The AMS is a generic framework for the implementation of the DPS. It lets sensor nodes maintain a set of candidate models that are periodically updated and evaluated. The set of candidates may include both different models, as well as several instances of the same model, corresponding to different parameter sets. Using an adequate performance measure, a sensor node can then periodically select the best performing model and thus adapt to changing data dynamics. We evaluated the performance of the AMS using two different sets of candidate models, namely *autoregressive prediction models* and *exponential smoothing prediction* models. In both cases, the AMS provides for nearly the same communication savings achievable with the optimal a posteriori model choice [116]. In addition, we provide an implementation of the AMS as a TinyOS library, for which we validated its performance on a small-scale deployment.

The remainder of this chapter is structured as follows: first, we provide a detailed description of the characteristics of the DPS in section 3.1. We describe related work in section 3.2, and present our instantiation of the DPS using the LMS adaptive filter in section 3.3. We introduce the AMS and provide the related experimental evaluation in sections 3.4 and 3.5, respectively. Finally, we report on our experiences in running the AMS on a small-scale lab deployment in section 3.6. Section 3.7 closes with a brief summary of this chapter.

Most of the contents of this chapter are also reported in [116, 166]. The design of the AMS and its implementation based on autoregressive models is the outcome of joint work with Yann-Aël Le Borgne and Gianluca Bontempi of the Université Libre de Bruxelles.

3.1. The Dual Prediction Scheme (DPS)

In typical WSN deployments, sensor nodes collect and report samples of a physical variable at regular time intervals [19, 30, 71, 121, 139, 185]. Thus, each sensor on a node captures a time series representing the development in time, at the location of the node, of the sensed variable. Since the values of a physical phenomenon, like air temperature or humidity, typically do not vary at random over time, successive elements of the captured time series are likely to be correlated. Therefore, it is often possible to derive adequate time series forecasting models that can be used to estimate, given a set of past observations, the values of the observed physical variable one or more time steps ahead [97, 114, 116, 122, 144, 145, 166, 194]. As discussed above, if a sensor node and the sink (to which the node reports its data) share the same prediction model, they can both compute the same estimations of the upcoming sensor readings. Since the sensor node also holds the collected samples, it can compute, after the i-th sampling operation, the actual estimation error e_i. Clearly, e_i represents the accuracy with which the sink can "reconstruct" the current sensor reading, even if does not receive any notification from the node. Thus, if e_i does not exceed a given *error budget* or *error threshold* e_{max}, no communication between the node and the sink is needed. On the other hand, if $e_i \geq e_{max}$ the node must send the sink a correspondent notification. This message typically includes the actual current reading and, possibly, the information necessary to update the prediction model at the sink. The process is then repeated for the successive sampling operations.

This data reporting strategy, which we refer to as the dual prediction scheme or DPS [166], can significantly reduce the amount of data communication between the node and the sink. At the same time, the DPS can guarantee the estimation error relative to each data sample to be within the interval $(-e_{max}, +e_{max})$.

An important assumption the DPS relies upon is that sensor nodes will typically collect sensor readings at a rate that is higher than strictly required for complying with the application requirements. Under such circumstances, the DPS can provide for communication savings since it can detect possibly existing redundancy in the collected samples. In many WSN deployments the temporal sampling rate is fixed a priori on the basis of empirical considerations and other requirements, such as network lifetime [10, 19, 30, 71, 121, 185]. Irrespectively of the applica-

tion scenario, most deployments will set the sampling rate high enough to avoid loosing important features that may show up unexpectedly in the data. Furthermore, the sampling rate is usually constant over time and equal for all the nodes in the network, even though the characteristics of the observed phenomenon may vary over both time and space. Therefore, it is reasonable to assume that the collected samples contain redundant information, at least over some intervals of time and across some sectors of the network. This redundancy can be eliminated by letting sensor nodes wake up and collect data at a lower frequency. However, this may cause a degradation of the accuracy of the collected data without providing for significant energy savings. In particular, we should recall that the energy consumption of many real-world sensors is significantly smaller than the energy required for communication, as also discussed in section 2.2.1. In these cases, and considering that in real deployments sensor nodes are likely to be required to wake up regularly anyway (e.g., to maintain time synchronization), assuming that a sensor operation is performed at each wake up is expected to have a negligible impact on the overall energy budget of a node.

Data collection based on the DPS also assumes reliable communication between the node and the sink. Indeed, if the node sends a notification that does not reach the sink, the latter erroneously considers its current data estimation to be within the error threshold e_{max}. In typical WSN deployments communication links may often be unreliable and messages losses can and do occur [19, 185]. In this case, as we also point out in [166], including a sequence number in each message can at least make the sink recognize that one or more notifications from the node have been lost. Thus, if the sink detects a jump in the sequence number of the messages received from a specific node, it can possibly start a dedicated procedure to recover the missing samples, or simply tag the affected estimations as potentially unreliable. Additionally, setting a limit to the maximum number of consecutive communication suppressions may also enable a timely detection of message losses. Indeed, a node may be forced to send a notification each time a watchdog timer T_w expires, irrespective of the actual value of the prediction error. If the sink does not receive any messages as its timer T_w expires, it assumes that a notification has been lost or the node is currently unavailable. For the remainder of this chapter, we assume the use of both sequence numbers and watchdog timers to be a sufficient countermeasure to cope with possible message losses. The proper value of the timer

T_w depends on the specific application, but it is assumed to be large enough so as to have only a negligible influence on the performance of the DPS.

Assuming reliable communication, although with the limitations discussed above, the actual communication savings achievable using the DPS depend on the ability of the chosen prediction model to estimate future sensor readings. The predictive ability of a model may depend on several factors, and it is usually hard to select a priori the optimal prediction model for a specific forecasting task [64, 65, 122, 134]. In the context of the DPS, the predictive ability may depend on the nature of the observed phenomenon, the error threshold e_{max}, or the sampling rate. Furthermore, the same model may show different performance if applied to distinct segments of the same time series or to series captured by neighboring nodes. Therefore, adapting the chosen prediction model to the actual collected data is instrumental in improving performance of the DPS. We provide more quantitative considerations on the issues of model choice and adaptation in section 3.1.1 below, after having introduced some basic notions on prediction models and the necessary mathematical notation.

3.1.1. Prediction Models and Parameter Estimation

Let $\mathbf{X} = \langle X_0, X_1, X_2, \ldots \rangle$ be the time series representing the sequence of sensor measurements $X_k \in \mathbb{R}$ collected at time instants $k = 0, 1, 2, \ldots$, $k \in \mathbb{N}$, and let $\mathbf{X}_{[\mathbf{0:k}]} = \langle X_0, X_1, X_2, \ldots, X_{k-1}, X_k \rangle$ be the sequence of observations from time 0 up to time instant t.[2] Using a prediction model $h(\mathbf{X}_{\mathbf{h,k}}, \boldsymbol{\theta}_{\boldsymbol{h,k}})$ it is possible to compute an estimate $\hat{X}_{k+1}$ of the upcoming time series element X_{k+1} as:

$$\hat{X}_{k+1} = h(\mathbf{X}_{\mathbf{h,k}}, \boldsymbol{\theta}_{\boldsymbol{h,k}}). \tag{3.1}$$

A prediction model $h(\mathbf{X}_{\mathbf{h,k}}, \boldsymbol{\theta}_{\boldsymbol{h,k}})$ is a mapping that takes as input a subset $\mathbf{X}_{\mathbf{h,k}}$ of the past time series elements, and a vector of N_θ model parameters $\boldsymbol{\theta}_{\boldsymbol{h,k}} = (\theta_1, \theta_2, \ldots, \theta_{N_\theta})$, with $N_\theta \in \mathbb{N}^+$. We stress the dependency of vectors $\mathbf{X}_{\mathbf{h,k}}$ and $\boldsymbol{\theta}_{\boldsymbol{h,k}}$ on the model h and on the time instant k by means of the subscript (h, k).

To clarify the role of the vectors $\mathbf{X}_{\mathbf{h,k}}$ and $\boldsymbol{\theta}_{\boldsymbol{h,k}}$, let us consider a

[2] If ΔT is the sampling interval and sampling starts at time $t = 0$, sensor readings are collected at instants $t_k = k\Delta T$, $k = 0, 1, \ldots$. Since the actual value of ΔT is irrelevant at this point, we refer to the sampling instants as $k = 0, 1, 2, \ldots$, for simplicity.

simple example of time series forecasting based on linear regression. Assuming that a time series $\mathbf{X}$ evolves linearly with the time k, a forecast of the element of the time series at time step $k+1$ can be computed using the following equation:

$$\hat{X}_{k+1} = a + b(k+1). \tag{3.2}$$

In equation 3.2 we assume that the parameters a and b have been estimated using a regression procedure over the past N elements of the time series, i.e., minimizing the sum of the square errors (least-square criterion):

$$(a_k, b_k) = argmin_{(a,b)} \sum_{i=0}^{N-1} (\hat{X}_{k-i} - X_{k-i})^2. \tag{3.3}$$

We emphasize the time-dependence of parameters a and b by the use of the subscript k. Using Equation 3.3, the parameter vector $\boldsymbol{\theta_{h,k}} = (a_k, b_k)$, is estimated using the past N elements of the series up to time instant k and therefore $\mathbf{X_{h,k}} = \mathbf{X_{[k-N+1:k]}}$. We can thus rewrite equation 3.2 in the form of equation 3.1 as:

$$\hat{X}_{k+1} = h(\mathbf{X_{h,k}}, \boldsymbol{\theta_{h,k}}) = h(\mathbf{X_{[k-N+1:k]}}, (a_k, b_k)), \tag{3.4}$$

for which we know h to be a linear model and the parameters (a_k, b_k) to be estimated using the least-square criterion as in equation 3.3. In general, vectors $\mathbf{X_{h,k}}$ and $\boldsymbol{\theta_{h,k}}$ are modified at each time instant k to take into account the newest collected element of the time series. From now on, we will refer to a change in $\mathbf{X_{h,k}}$ or $\boldsymbol{\theta_{h,k}}$ as a *model update*.

The values of the parameters a and b could also be fixed a priori, possibly on the basis of some historical data or other available side information. However, this choice would make the model unable to adapt to the actual collected data and thus possibly hamper its predictive ability. Therefore, choosing a model to perform time series forecasting requires determining both a model "template" (constant, linear, quadratic), which fixes the number and the nature of the model parameters, and an adequate procedure to compute and update these parameters. In the context of the DPS, the nodes can store the last collected elements of the time series and use them to update the model parameters. To this end, the parameter estimation procedure must be executable on resource-constrained wireless sensor nodes. This requirement may disqualify several classes of models as potential candidates

to implement the DPS.

For instance, ARIMA[3] models represent a widely used class of prediction models. They have been successfully adopted to describe a large variety of phenomena, from financial to environmental time series [122]. Additionally, the Box-Jenkins method provides an analytical procedure to estimate the optimal ARIMA model parameters for a given time series [28]. However, this procedure requires a computationally expensive estimation of the sample autocorrelation and partial autocorrelation functions of the series, which in turn can only be reliably estimated from a large (typically > 50) number of samples [28]. Using ARIMA models to implement the DPS would thus require running the Box-Jenkins selection procedure on sensor nodes. However, since sensor nodes typically feature only few kilobytes of RAM and poor computational capabilities [154], this is actually impractical. Furthermore, the use of sophisticated (and expensive) predictors like ARIMA models does not guarantee for the computation of more reliable predictions [122].

Besides ARIMA models, there exist of course several other generic model "templates" that can be used to perform time series prediction [64, 122, 134]. But selecting an appropriate model for a given forecasting task is a not trivial procedure. A popular survey on time series predictions, written back in 1985, concluded that the need to "*establish some basis for choosing among these and other approaches to time series forecasting*" was one of the major challenges for future research [64]. An analogous statement concludes, twenty years later, a revised and extended version of that survey paper [65].

In several application scenarios model selection is based on a priori knowledge or the judgment of an expert [8, 122]. Automatic selection procedures have also been investigated [8, 122, 124, 184] but, as mentioned above, these procedures may be computationally expensive and require large sets of historical data to be available. In the context of WSNs, the computation of model parameters could also be delegated to a central server, once a sufficient amount of sensor readings have been collected. The server could then notify its model choice back to the nodes and run the DPS as usual. However, this centralized solution does not scale well as the network size increases. Furthermore, the initial model choice may need periodical refinements to take into account changes in the data, which cannot be done from the predicted values

[3] Auto-Regressive Integrated Moving Average.

but which requires that the server is supplied with "fresh" sensor readings. This, in turn, would require the nodes to periodically transmit a potentially large amount of consecutive sensor readings. Therefore, we believe that performing model selection on the nodes is, in spite of the above sketched challenges, the most reasonable approach to provide for efficient implementations of the DPS.

In the light of these considerations, it is not surprising that existing implementations of the DPS basically differ in terms of the model used to perform forecasting, and the related procedures used to update model parameters. In the following section 3.2, we discuss some interesting implementations of the DPS that have been presented within the WSN research community. We then present our own approaches in sections 3.3 and 3.4.

3.2. Related Work

Several authors within the WSN research community considered the potential of prediction-based techniques to optimize data collection. In [144, 145] Olston *et al.* propose one of the first, and simplest, implementations of the DPS. To this end, they leverage an approach that had been originally developed to speed up data retrieval from remote databases, known as *quasi-copies* [6]. Quasi-copies are replicas of data stored in a remote database that are cached at a user's site. This replicas are allowed to deviate from the true, centrally stored values in a controlled fashion. In particular, the cached copies are guaranteed to lie within a given range from the actual values. In the context of WSNs, the quasi-copies approach can be implemented by installing appropriate data filters at each sensor node. These filters drop all the readings being $\pm e_{max}$ off the last sensor measurements that has been sent to the sink, where e_{max} is the tolerated error on the collected data. Thus, the resulting data copy at the sink consists in a piecewise constant approximation of the actual time series observed at the nodes. This approach actually consist in performing time series prediction using a "naïve", or *constant* model [122], which just provides the last recorded (i.e., sent) measurement as the forecast for the next sensor readings. In [144] Olston *et al.* also propose to make the actual error threshold e_{max} adapt to the current data transmission costs and to the individual *data change rates* experienced at each node. Lazaridis and Mehrotra [114] also investigate using the constant model to provide for prediction-based data

collection in WSNs.

As we show in section 3.5, the constant model, hereinafter also referred to as CM, performs surprisingly well in practical settings. Furthermore, it actually constitutes a sort of minimal effort implementation of the DPS and thus can be used as the default model choice. However, the simplicity of the CM often does not allow to exploit the temporal correlation of successive elements of a time series fully. Therefore, several authors proposed using more sophisticated prediction techniques to implement the DPS.

For instance, Jain *et al.* [97] propose an implementation of the DPS based on the Dual Kalman Filter (DKF) architecture. In the DKF approach each remote source (i.e., sensor node) involved in a specific sensing task runs an instance of a Kalman filter and performs linear prediction on smoothed sensor readings. As in the "usual" DPS, data sources send updates to a central server only when the prediction error exceeds a pre-specified error threshold. The central server holds as many Kalman filters as the number of remote sources. In this way, the server is able to mirror the filters installed at the data sources and thus reconstruct the values observed at each sensor node using either the received actual data, when available, or the computed predictions. In order to use the Kalman filter for data streams prediction given a sequence of noisy observations, a model of the observed phenomenon must be provided to the filter (obviously, both server and nodes must feed the Kalman filter with the same model to be able to work coherently), i.e., the statistical properties of both the observed phenomenon and the noise process must be known a priori [101]. This limits the applicability of this method as a general framework for sensor data forecasting in WSNs, since a priori knowledge of the observed time series is often unavailable or unreliable.

In our own work [166], discussed also in the next section 3.3, we propose an implementation of the DPS that allows to overcome the problem of defining a priori knowledge on the signals of interest. To this end, we suggest using (linear) adaptive filters, which are able to learn signal statistics on the fly, and can continuously and autonomously adapt to changes [82]. In particular, we instantiate the DPS using the lightweight LMS adaptive predictor, which basically performs linear regression over the past n readings available at the sink. This approach performs well on real-world sensor data, and requires only few memory and computational resources. Its major drawback lies in the need for

defining the number of samples needed, i.e., the actual value of n, a priori. One thus needs to have an adequate method to estimate this model parameter on the fly, i.e., as data collection is performed.

Automatic estimation procedures to determine model parameters are also missing in the approach presented by Tulone and Madden [194]. In their work, the authors instantiate the DPS using autoregressive (AR) models, but do not provide a method for on-line, automatic selection of the autoregressive order of the model (see also section 3.5 for more details on autoregressive models). On the other hand, they propose an interesting policy for choosing an adequate point in time to update the autoregressive coefficients of the model. To this end, they define, besides the error budget e_{max}, a second threshold δ ($0 < \delta < e_{max}$). As soon as the number of occurrences in which the prediction error is bigger than δ exceeds a third given threshold a, a model update is performed. In our work, we address the issue of automatic estimation of model parameters by introducing a generic framework for model selection, as described in section 3.4.

The above discussed techniques mainly focus on time series prediction as a mathematical tool to perform temporal sensor selection in WSNs. Other approaches leverage prediction-based data collection that also takes into account the spatial dimension. For instance, Goel and Imilienski [69] suggest to visualize a snapshot of the sensor readings in the network as an optical image. With this premise, the authors suggest to resort to the MPEG[4] standard for video compression to predict future sensor readings. This requires a base station to first collect enough sensor readings from the sensors to generate a suitable prediction-model, which is valid over a limited time interval. The model is then propagated to the sensor nodes, which send their readings only if they significantly differ from those predicted by the model. A similar, model-driven approach is proposed by Deshpande *et al.* [46]. In this case, a spatio-temporal prediction model is learnt from historical data and is then used to estimate sensor readings in the current time period. The estimation computed by the model can possibly be refined by interrogating the sensor network for some specific current readings. Guestrin *et al.* [76] propose to build a model of the data in the network using *kernel linear regression* and let the nodes transmit only significant changes in the model coefficients instead of raw data. In [115], principal component analysis (PCA) is used to identify minimal subsets

[4] Moving Picture Experts Group [117].

of nodes whose readings allow to predict the values sensed at any node in the network. The subsets then provide sensor readings in a round robin fashion, so that the overall resource consumption is reduced and balanced across the nodes.

This chapter focuses on the temporal sensor selection problem only and thus we do not address prediction-based techniques working in the spatial domain. Notwithstanding, we investigate the spatial sensor selection problem in chapter 4.

3.3. An Implementation of the DPS Based on the LMS Adaptive Filter

To enable a lightweight and flexible implementation of the DPS, we propose to make use of the least-mean-square (LMS) adaptive filter [82, 83, 166]. The LMS has very low computational overhead and memory footprint, and can therefore be easily implemented on sensor nodes. Furthermore, it does not require a priori knowledge of the statistical properties of the observed signals [82, 83]. In particular, the LMS can adapt on the fly to the actual signal dynamics and can thus be applied to a variety of real-world phenomena. Moreover, nodes do not need to be assisted by a central entity to run the LMS, since no global model parameters need to be defined. Its only drawback, which it shares with other DPS instantiations, is that each node's individual model parameters can really only be set optimally during data collection (in the case of LMS this is the filter length N_{LMS} and its step-size μ). We will address this later (in section 3.4) by proposing a novel generic online parameter estimation and model selection framework, called AMS.

In the following, we first introduce basic notions on adaptive filters and the LMS, and show how the latter can be used to perform time series prediction. We then discuss the details of our implementation of the DPS using the LMS, and finally present the related experimental evaluation based on several data sets collected in real WSN deployments.

3.3.1. Adaptive Filters and the LMS Algorithm

In order to formally define the LMS, we first introduce the generic structure of a linear adaptive filter. To this end, the samples of a physical variable X are assumed to be available at discrete time instant

$t_k = k\Delta T$, $k = 1, 2, \ldots$. Thus, every ΔT time units (e.g., seconds) a new sample X_k (short for X_{t_k}) of the signal is available. At each time step t_k, a linear adaptive filter of filter length N_{LMS} takes as input the newest collected sample of the signal X, along with the precedent $N_{LMS} - 1$, and computes the filter output Y_k as:

$$Y_k = \sum_{i=0}^{N_{LMS}-1} w_{i+1,k} X_{k-i}. \tag{3.5}$$

Setting a weights vector $\underline{w}_k = w_{1,k}, w_{2,k}, \ldots, w_{N_{LMS},k}$, and defining the sample vector $\underline{X}_k = X_k, X_{k-1}, \ldots, X_{k-N_{LMS}+1}$, we can rewrite equation 3.5 above in a more compact form as the scalar product:

$$Y_k = \underline{w}_k \cdot \underline{X}_k^T, \tag{3.6}$$

where $(.)^T$ is the transposition operator. The output of the filter is thus the linear combination of the last N_{LMS} samples of the input signal X, each one of them being weighted by a time-varying filter coefficient $w_{i,k}$. The filter output Y_k is then compared to a reference value Y_k^d (d for *desired*). Y_k^d represents the sample, at time t_k, of a reference signal Y^d to which the filter tries to adapt. In other words, the filter performs optimally if $Y_k = Y_k^d$,$k = 1, 2, \ldots$. Thus, the error signal $e_k = Y_k^d - Y_k$ at time instant t_k is fed back to the filter and used to update the filter weights. Figure 3.1(a) shows the generic structure of an adaptive filter working along the rationale described above.

The weights vector $\underline{w}_k$ is modified at each time step k according to a given optimality criterion, which is typically the minimization of the mean square error (MSE), i.e., the average power of the error signal e. Without going into details, we point out that the choice of the MSE as the optimality criterion implies that the error function $J(\underline{w})$, which describes the dependency of the MSE on the filter weights $\underline{w}$, is a quadratic function. Thus, $J(\underline{w})$ has a unique absolute minimum point $\underline{w}^{opt}$, i.e., a unique optimal solution that minimizes the MSE [83]. The filter weights are updated at each step k in order to iteratively approach this minimum point. The error e_k gives the adaptation algorithm a measure of the extent of the correction that needs to be applied to the filter weights in order to reduce, at the subsequent step $k+1$, the expected error power $E\{e_{k+1}\}$. If the statistics[5] of the involved signals

[5] The most important values are: the autocorrelation matrix of the input signal; the cross-correlation vector of the input; and the reference signal.

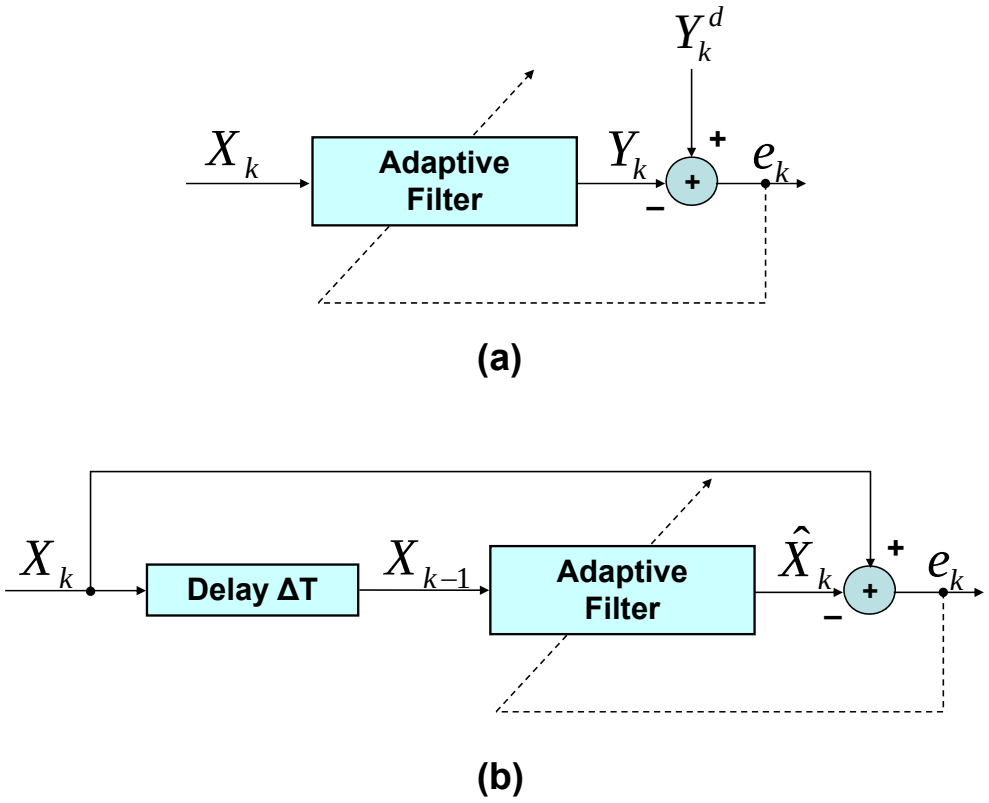


Figure 3.1.: Adaptive filter: (a) generic structure, (b) as a prediction filter.

were stationary and known a priori, the set of optimal filter weights $\underline{w}^{opt}$ that minimizes the MSE could be computed through the Wiener-Hopf equation [82, 146]. In practical scenarios, however, a priori knowledge of the signal statistics may be unavailable or unreliable. Adaptive filters can learn these statistics from the data and adapt to their changes by updating the filter weights $\underline{w}$. In this sense, adaptive filters provide a *tracking* capability, since they are able, in a non-stationary environment, to track variations in the statistics of the input data, provided that these variations are sufficiently slow [82].

Adaptive filters are usually categorized depending on the specific method used to update the filter weights, and a large number of such adaptive algorithms is available in the literature [82]. The choice of one method over another is determined by the trade-off among different factors, like rate of convergence, robustness, computational complexity, or numerical properties. The LMS is one of the most successfully applied adaptive algorithms. Despite its simplicity, it provides for good performance in a wide spectrum of applications [83]. The equations that define the LMS algorithm are reported in table 3.1. The parameter μ regulates the convergence speed of the weights adaptation procedure,

as we detail below. Like any other adaptive filter, the LMS algorithm can be used to perform prediction when the general filter structure in figure 3.1(a) is refined as in the predictive structure of 3.1(b). Making a predictor out of an adaptive filter requires delaying the current sample X_k by one step and using it as the reference signal Y_k^d. The filter then computes an estimation $\hat{X}_k$ of the input signal at step k as a linear combination of the N_{LMS} previous readings $\{X_{k-1}, X_{k-2}, \ldots, X_{k-N_{LMS}}\}$:

$$\hat{X}_k = \sum_{i=1}^{N_{LMS}} w_{i,k} * X_{k-i}. \tag{3.7}$$

The prediction error is then computed and fed back to adapt the filter weights. The characteristics of the adaptation process can be controlled through two parameters: the *step-size* μ, that appears in the weight update equation in table 3.1, and the *filter length* N_{LMS}. Using the notation introduced in section 3.1.1, we thus have:

$$\hat{X}_{k+1} = h(\mathbf{X}_{[\mathbf{k-1,k-N_{LMS}}]}, (w_{0,k}, w_{1,k}, \ldots, w_{N_{LMS},k}, \mu, N_{LMS})), \tag{3.8}$$

where the model h is the LMS filter.

The step-size μ is a critical parameter for the practical implementation of the LMS, since it tunes the convergence speed of the algorithm. The theoretical convergence analysis of the LMS is still a field of active research and a direct mathematical theory for its stability is still far from being complete [83]. Indeed, even though the filter was introduced as early as 1959, only very recently the first rigorous equation for a necessary stability bound on the step-size parameter μ of the LMS has been provided [32]. Nonetheless, there also exists a practical criterion for a straightforward computation of the value of μ from a small set of observations, as we point out in the next section 3.3.2.

The number of filter weights, normally referred to as the *filter length* N_{LMS}, mainly determines the computational load and memory footprint of the filter. From the equations reported in table 3.1, and recalling that $\underline{w}$ and $\underline{X}$ are vectors with N_{LMS} elements, it follows that the LMS algorithm requires $2N_{LMS}+1$ multiplications and $2N_{LMS}$ additions per iteration. In particular, N_{LMS} multiplications and N_{LMS} additions are required for computing the filter output Y_k, while N_{LMS} additions and $N_{LMS}+1$ multiplications are required to update the filter coefficients $\underline{w}_k$. Thus, in order to keep the computational load of the

filter low, the number of weights N_{LMS} must be kept as low as possible. Our experimental evaluation shows that the filter performs well even with $N_{LMS} = 4, \ldots, 10$. We also point out that increasing the value of N_{LMS} does not necessarily improve the performance of the filter. In particular, increasing N_{LMS} above a theoretically determinable threshold value N_{LMS}^{opt} will even result in performance losses [83].

For further details about the characteristics and application fields of the LMS algorithm we refer to [83]. In the following, we will explain how this algorithm can be used to provide for an efficient implementation of the DPS.

Table 3.1.: The LMS Algorithm

$Y_k = \underline{w}_k \underline{X}_k$	Filter output
$e_k = Y_k^d - Y_k$	Error signal
$\underline{w}_{k+1} = \underline{w}_k + \mu \underline{X}_k e_k$	Weights adaptation

3.3.2. Implementation of the DPS Using the LMS

As we show in the next section 3.3.3, using the LMS for implementing the DPS allows the algorithm to significantly reduce the amount of data a node is required to report to its sink. At the same time, the collected data can be guaranteed to lie within a given maximum error budget e_{max}. This reduction is achieved by letting the node switch as frequently as possible from a *normal* operational mode to a so-called *stand-alone* mode, in which the node does not need to report sensor readings to the sink. In order to be able to run the prediction algorithm, the node needs to go through an *initialization* phase. These three basic states of node operation are described in the remainder of this section.

Initialization Mode. When the node starts collecting and reporting data, it runs in initialization mode. During this phase, no prediction is performed and the both the node and the sink use the available actual samples to compute an estimation of the step-size μ. To ensure convergence, the step-size μ must satisfy the following condition [83]:

$$0 \leq \mu \leq \frac{1}{E_X} \tag{3.9}$$

where E_X indicates the mean input power computed as:

$$E_X = \frac{1}{M} \cdot \sum_{k=1}^{M} |X_k|^2 \tag{3.10}$$

and M is the number of samples used to train the filter [82]. Since the input mean power E_X is time-varying, an approximation $\bar{E}_X$ can be computed over (at least) the first N_{LMS} samples and used to compute the upper bound in inequality 3.9 above. In practical applications, the step-size μ can be assigned a value about two orders of magnitude smaller than this bound [135]. To take into account changes in the signal dynamics, the node should periodically recompute the value of μ, and communicate it to the sink. Furthermore, including a mechanism to allow for an on-line estimation of the optimal filter length N_{LMS} would also be appropriate. Such a mechanism could be implemented using the AMS framework discussed in section 3.4.

Once the initialization phase is completed, both the node and the sink start performing prediction on the collected readings and operate in either *normal* or *stand-alone* mode, as explained below.

Normal Mode. When working in normal mode, both the node and the sink use the last N_{LMS} readings to compute a prediction for the upcoming measurement, and update the set of filter coefficients $\underline{w}$ on the basis of the actual prediction error, using the equation given in table 3.1. The default start value for the filter coefficients is assumed to be $\underline{w}[0] = \underline{0}$. Unlike other adaptive algorithms, the LMS ensures that multiple instances of the filter fed with the same sequence of data and sharing the same set of initial weights $\underline{w}[0]$ (and, of course, the same values for N_{LMS} and μ), compute the same set of filter coefficients and thus the same predictions at each time instant k.

As long as the prediction error exceeds the user defined error budget e_{max}, the node keeps working in normal mode, thus collecting and reporting its readings to the sink. When the prediction error drops below the threshold e_{max} for N_{LMS} consecutive steps, the node switches to *stand-alone* mode. As long as the node remains in the *normal* mode, the sink lets the prediction filter run over the received sensor readings, in order to update the filter weights $\underline{w}$ coherently with the node.

Stand-Alone Mode. When working in *stand-alone* mode, the node keeps collecting data and computes the prediction at each time step. As long as the prediction error remains below the given threshold e_{max}, the node discards the reading and feeds the filter with the prediction $\hat{X}_k$ instead of with the real data X_k. This ensures that the state of the filter at the node remains consistent with the state of the filter at the sink. Feeding the filter with its own prediction causes the prediction error to be zero and thus the filter weights to be left unchanged. This is another advantage of using this technique: while staying in *stand-alone* mode, the node can omit updating its weights, thus saving half of the computational overhead. If at time instant k the node observes that the prediction error exceeds the threshold e_{max}, it will report the reading X_k to the sink and switch back to *normal* mode. While the node operates in *stand-alone* mode, the sink interprets the lack of readings from the node as successful predictions, i.e., that the predicted readings are a good enough approximation of the real readings at the node, and thus continues to use the values from its own prediction filter.[6]

Figure 3.2 illustrates how our scheme works. We let our algorithm (with $N_{LMS} = 5$ and $\mu = 10^{-5}$) run on a set of temperature readings obtained from a real world sensor [19], as shown in Figure 3.2(a). Figure 3.2(b) shows a detailed view of the outlined area in subfigure (a), with an overlapping plot of the corresponding filter output. Subfigure (c) shows the prediction error of the data points in subfigure (b), including highlights (with a cross) for those readings that the node effectively needs to report to the sink in order to guarantee an accuracy e_{max} of $\pm 0.5°C$. We see that as soon as the error exceeds the given threshold, the corresponding sensor reading is sent to the sink and the filter restarts adapting to the real data, thus causing the prediction error to decrease. As soon as the error remains below $\pm 0.5°C$ for at least $N_{LMS} = 5$ readings, the node stops reporting data (i.e., switches again to *stand-alone* mode).

We should also point out that an *outlier detection*[7] mechanism could be easily embedded into our scheme. Since the occurrence of outliers may disturb the operation of any prediction filter, it is good practice to include some automatic procedure for their detection. For instance, an

[6] Obviously, the absence of a message might also be due to a crash or battery failure at the node, so we assume that nodes send readings or at least status messages at regular intervals, so that the sink can easily recognize the absence of an expected message. To this end, a loose synchronization between the sink and the nodes is required.

[7] An outlier is a "*data value that is unusually large or small*" [122, page 609].

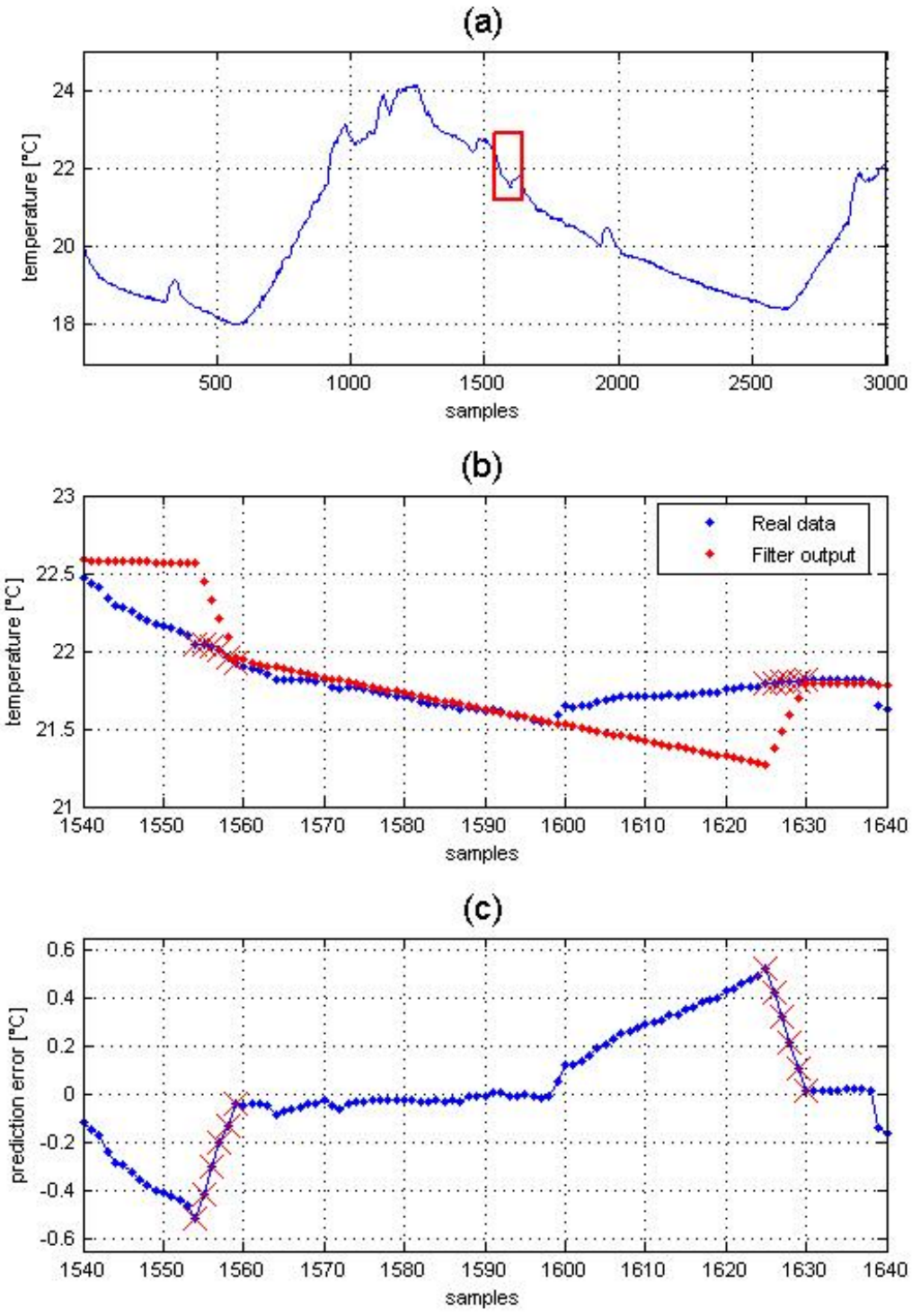


Figure 3.2.: The LMS at work: (a) real sensor readings, (b) real and predicted sensor readings (close-up of the framed area in subfigure (a)), (c) prediction error.

adequate threshold may be defined either by the user or by the node itself (e.g., as a multiple of the mean error). A sensor reading whose corresponding prediction error is larger than this threshold could thus be classified as an outlier and discarded. In this case, the discarded data can be replaced by the corresponding prediction.

3.3.3. Experimental Results

Finally, we present the experimental results obtained by applying our LMS-based DPS to several publicly available data sets. In particular, we used temperature readings collected within the Intel Berkeley Research Lab deployment [19]. These data sets include humidity, temperature, light and voltage readings collected, once every 31 seconds, by 54 Mica2Dot sensor nodes [43]. The nodes were deployed across a floor of the Intel Lab building and collected sensor readings between February 28 and April 5, 2004.

Four our empirical study, we picked 4 of these 54 motes, namely motes 1, 11, 13, and 49, which were distributed in different sectors of the deployment area. We applied our scheme to the data reported by the temperature sensors of these four motes between March 6 and March 9, 2004. Since the main goal of the DPS is to reduce data communication between the node and the data sink, we use the number of updates the nodes send to the sink during operation as the performance metric of choice. We define this metric, named the *relative update rate*, as the ratio of the number of updates effectively sent when running the DPS to the number of updates that would have been sent by the default monitoring scheme. The results reported in this section have been obtained implementing the LMS-based DPS in Matlab (see section 2.2.3).

Figure 3.3 shows the relative update rate (in percentage) of mote 11 for three different parameter sets plotted over the error budget e_{max}. As we can see, a minimum accuracy of $0.5°C$ can be guaranteed while transmitting only about 10% of the collected sensor readings. This significant reduction in data communication is due to the remarkable tracking capability of the LMS algorithm. Moreover, no significant changes in the performance are observed when varying the number of filter weights from $N_{LMS} = 4$ to $N_{LMS} = 10$. Since the number of operations to be performed at each time step grows proportionally[8] with

[8] We recall from section 3.3.1 that the computational cost per iteration of the LMS algorithm is

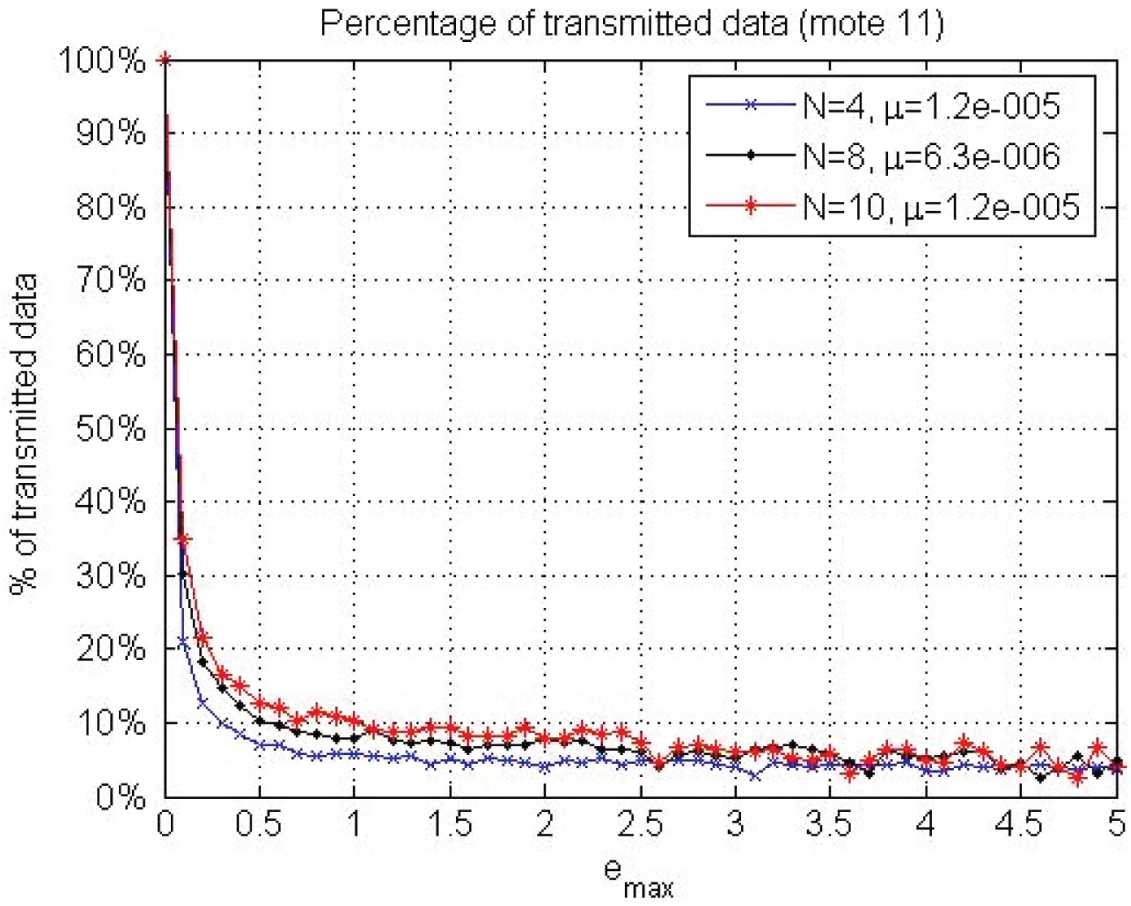


Figure 3.3.: Relative update rate for mote 11.

N_{LMS}, this value should be kept as small as possible. The tested values of N_{LMS} allow us to keep the computational overhead and memory footprint extremely low. For instance, with $N_{LMS} = 4$ the node must perform at most 17 operations each 31 seconds and needs to store only the last 4 sensor readings in addition to the 4 filter coefficients and the filter parameters.

We also obtained encouraging results for the data collected by other motes. For instance, figure 3.4 shows the relative update rate of mote 49. The small performance loss with respect to mote 11 is due to the fact that the samples collected by mote 49 are more spiky than those of mote 11. Following these abrupt changes requires the LMS to send more updates. Finally, figure 3.5 shows the performance obtained with two additional data sets, namely those collected by mote 1 and 13. Mote 1 is located far away from both mote 11 and 49, while mote 13 lies in the same room as mote 11. Also with these data sets we obtained very good results in terms of data reduction.

$4N_{LMS} + 1$ when the node operates in *normal* mode and $2N_{LMS}$ in *stand-alone* mode.

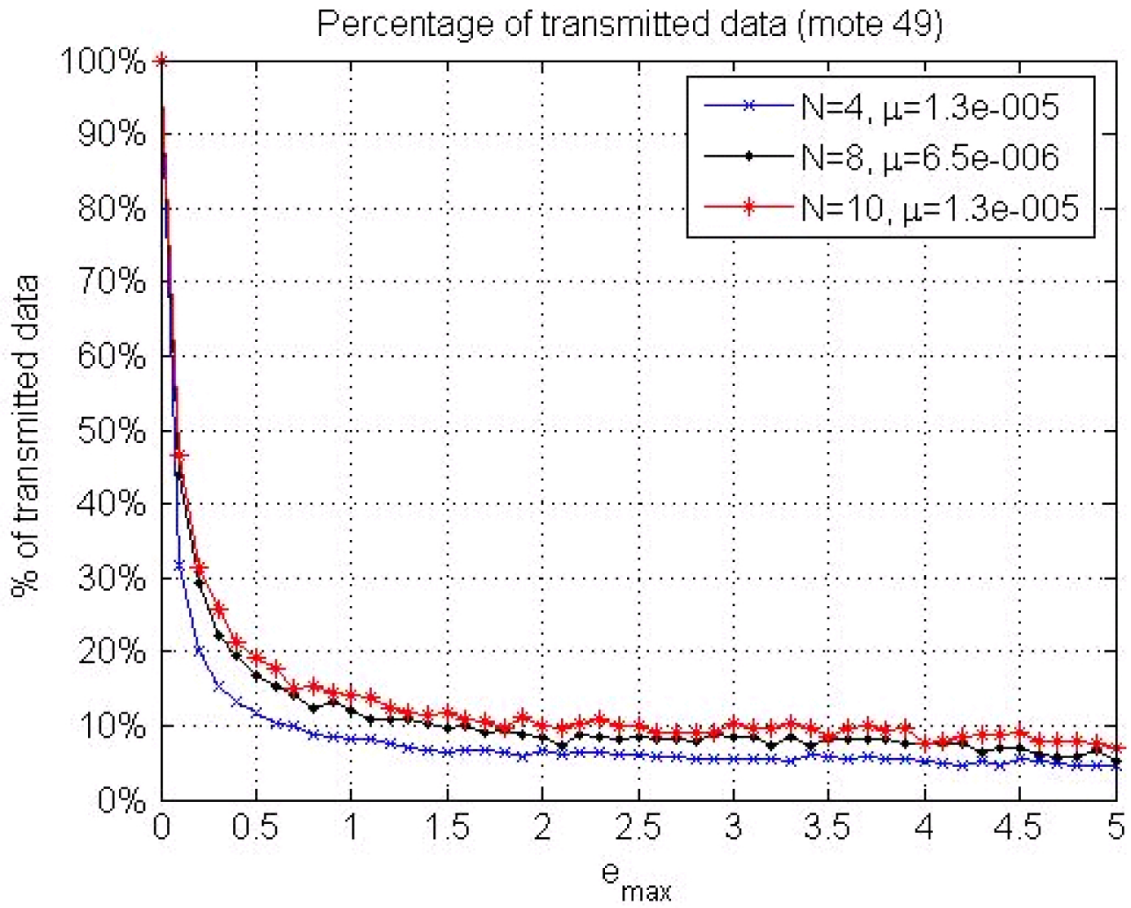


Figure 3.4.: Relative update rate for mote 49.

The above discussed results show the potential communication savings achievable using LMS as the prediction model of choice for implementing the DPS. As we also mentioned above, one drawback of this approach is the fact that the filter length N_{LMS} is fixed a priori although its optimal value can only be determined during data collection. To cope with this problem, common to other implementations of the DPS, we propose a generic framework for on-line parameter estimation and model selection, the AMS, which is described in detail in the next section 3.4.

3.4. Adaptive Model Selection (AMS)

In section 3.1 we discussed the challenge represented by the selection of an adequate model for supporting a given forecasting task. Furthermore, we stressed the importance of progressively refining the values of the parameters of the selected model in order to ensure its ability to follow changing signal dynamics. However, many of the approaches

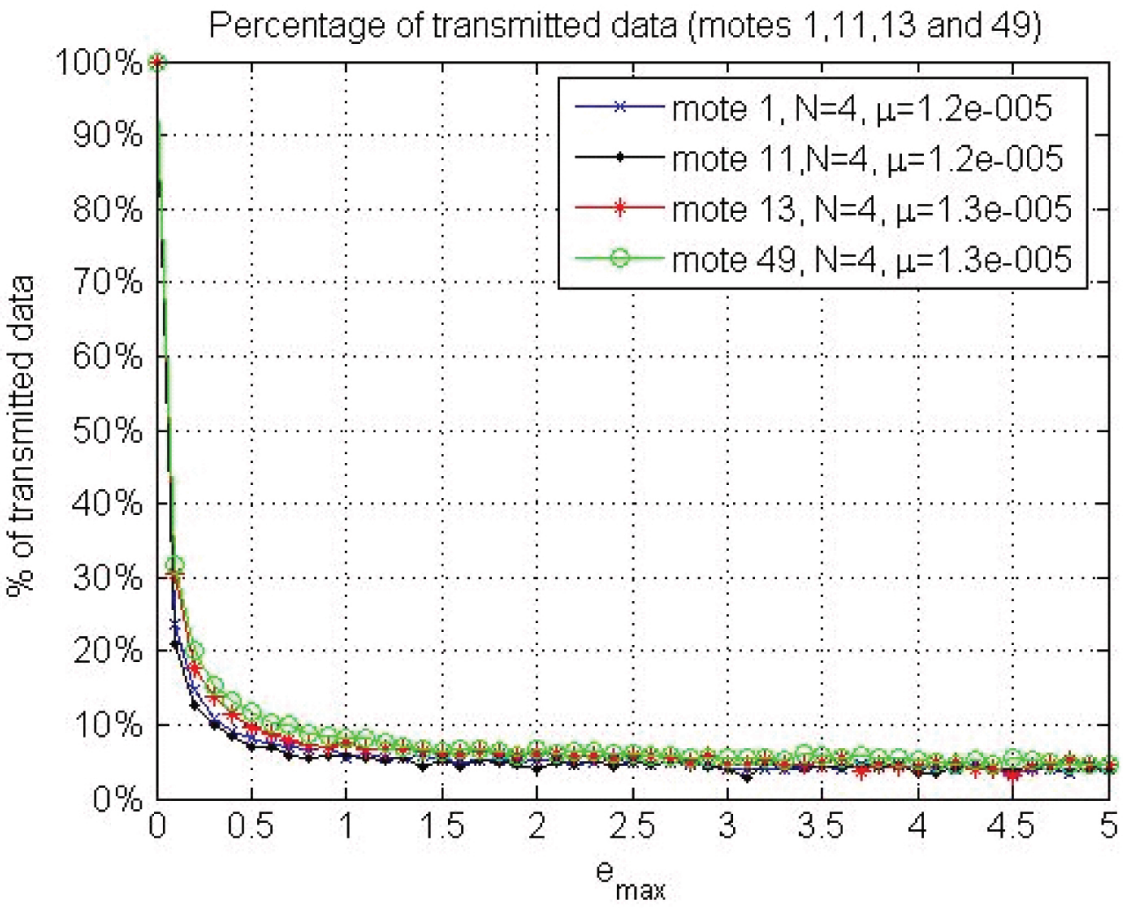


Figure 3.5.: Relative update rate for motes 1, 11, 13, and 49, for a filter length $N_{LMS} = 4$.

presented in section 3.2, including our own work on the LMS-based DPS, do not provide any parameter update procedure, or alternatively suggest methods that require collaboration with the sink or high computational costs and memory usage. Furthermore, none of the previously described DPS implementations provides a way to monitor the performance of the used prediction model. However, monitoring the communication gains achieved by the DPS using different models would allow one to correct a possibly inappropriate initial model or parameter choice on the fly.

To overcome these drawbacks, we designed a generic framework for online model selection. The rationale of our adaptive model selection (AMS) framework is to let the sensor nodes run a set of different prediction models and evaluate, for each of these models and at each sampling round, a quality measure that describes the efficiency of the model in the currently engaged DPS. In this way, each time data communication is required, the nodes can select the currently best performing model

out of the set of candidates and send it to the sink, which will use it to predict future sensor readings until the next update is received.

In principle, the AMS can be implemented with an arbitrary number of models of the same class or of different type. For instance, the set of candidate models could be represented by several LMS filters corresponding to different values of N_{LMS}. Or by the constant model, an instance of the LMS, and a few ARIMA models. However, the computational load and memory footprint of the AMS increase with the number and complexity of the models. Therefore, along with the general AMS framework presented in section 3.4.3 below, we also propose two specific implementations of an AMS-based DPS.

Our first prototypical implementation of the AMS restricts the set of candidates to the constant model (CM) and a few representatives of the *autoregressive* (AR) model class. This choice is mainly driven by the fact that AR models have been widely used to implement the DPS in WSNs. However, AR models may be difficult to compute and maintain, and thus only a few of them may be included in the set of candidate models [116]. We therefore propose a second, more lightweight and generic implementation of the AMS, suitable for currently available sensor networks platforms, that relies on *exponential smoothing* (ES) models. ES models guarantee good behavior on a number of different time series and are computationally cheap to maintain [64, 65]. Therefore, we propose this class of models as the most appropriate for implementing the AMS algorithm and we refer to this implementation as the ES-AMS. We provide a formal definition of both AR and ES models in section 3.5.1.

In the following, we first clarify the performance metric the AMS uses to select the best performing model out of the set of candidates. We then describe the *racing* mechanism, which can be used to discard poorly performing models from the set of candidates. Finally, we present the AMS algorithm and discuss some relevant implementation issues. The experimental evaluation of the AR-AMS and ES-AMS is then provided in the following section 3.5.

3.4.1. Performance Estimates

As sketched above, a sensor node running the AMS maintains a set of N_{AMS} *candidate* prediction models $\{h_n\} = h_n(\mathbf{X}_{h_n,k}, \boldsymbol{\theta}_{h_n,k})$, $1 \leq n \leq N_{AMS}$, which are possibly updated at each time instant k. For

each model h_n, a given quality measure is recursively estimated, and the model that optimizes this performance indicator is selected as the *current* model.

As we stated previously, the main goal of the DPS is to reduce the number of updates between a sensor node and the sink. To measure the performance of the DPS we thus resort to the *relative update rate*, as already done in section 3.3.3. The *relative update rate* is the ratio of the number of updates effectively sent when running the DPS to the number of updates that would have been sent by the default monitoring scheme. Let $U_{h_n,k}$ be the relative update rate for the model h_n at time k, where $U_{h_n,1} = 1$, $1 \leq n \leq N_{AMS}$. $U_{h_n,k}$ can be recursively computed as

$$U_{h_n,k} = \frac{(k-1) * U_{h_n,k-1} + 1}{k}, \tag{3.11}$$

if an update is needed at time k, or as

$$U_{h_n,k} = \frac{(k-1) * U_{h_n,k-1}}{k}, \tag{3.12}$$

otherwise. The relative update rate for the default monitoring scheme is 1, since it requires the transmission of all the collected readings, and thus any lower value indicates a gain in terms of data communication. We usually report the update rate as a percentage, thus $U_{h_n,k} = 1$ means that 100% of the collected samples have been actually transmitted to the sink.

Performance assessment in terms of update rate has been considered in several implementations of the DPS [97, 114, 166, 194]. However, we have been the first in suggesting to use this indicator to measure model performance on the node, in order to enable a flexible model choice [116].

In the context of the AMS it may also be useful to consider the size of a model update, in terms of number of parameters (and thus number of bytes) to be transmitted, as part of the performance indicator. Indeed, different models within the set of candidates may require a variable number of elements to be sent at each update. For instance, the default monitoring mode only requires sending the current sensor readings, while, in general, updating a model h_n requires sending both the input values $\mathbf{X}_{h_n,k}$ and the model parameters $\boldsymbol{\theta}_{h_n,k}$. To take into account the packet size of a single model update we introduce an alternative performance indicator, the *weighted relative update rate*, which

we define as follows:

$$W_{h_n,k} = U_{h_n,k} * C_{h_n}. \tag{3.13}$$

C_{h_n} is the ratio of the number of bytes required to send an update of model h_n to the number of bytes required to send an update in the default monitoring mode. We refer to C_{h_n} as the *model cost*. The weighted relative update rate $W_{h_n,k}$ measures the savings in terms of data rate for model h_n at time k with respect to the default monitoring mode. When the costs C_{h_n} of the models h_n differ significantly, the weighted relative update rate represents a "fairer" indicator of the actual performance of the individual models. When all models have the same cost, the weighted relative update rate and the relative update rate are equivalent.

3.4.2. Racing Mechanism

The number and type of models included in the set of candidates may vary depending on the application, and the available computational resources and memory resources. However, some of the models initially included in the set may turn out to perform poorly in terms of relative update rate or weighted relative update rate. Since maintaining these models wastes precious computational and memory resources, it is desirable to discard them as soon as possible from the set of candidates. In this context, the *racing mechanism* [124] offers an effective approach for the automatic identification of models that persistently perform poorly with respect to other candidates. The rationale of the racing mechanism is to determine, on the basis of hypothesis testing [80], what models among a set of candidates are significantly outperformed by others. For instance, let $h^* = argmin_{h_n} W_{h_n,k}$ be the model with the lowest relative data rate at time instant k among the set of candidate models, and let $\Delta_{(h_n,h^*),k} = W_{h_n,k} - W_{h^*,k}$ be the difference between the estimated weighted relative update rates of any model h_n and h^*. Relying on the Hoeffding bound [87], a distribution free statistical bound, the racing mechanism assumes with probability $1 - \delta$ that h^* truly outperforms h_n if

$$\Delta_{(h_n,h_n^*),k} > R\sqrt{\frac{ln(1/\delta)}{2k}}, \tag{3.14}$$

where R is the range taken by the random variable $\Delta_{(h_n,h^*)}$. Thanks to the lack of parametric assumptions, the Hoeffding bound requires no other information than the range of values taken by the random variables considered, which is known at the nodes. Indeed, since $0 \leq W_{h_n,k} \leq C_{h_n}$ and $0 \leq W_{h^*,k} \leq C_{h^*}$, it follows that $R = C_{h_n} + C_{h^*}$, and the bound for discarding model h_n is therefore given by:

$$\Delta_{(h_n,h^*),k} > (C_{h_n} + C_{h^*})\sqrt{\frac{ln(1/\delta)}{2k}}. \tag{3.15}$$

The racing mechanism allows to discard poorly performing models from the set of candidates among which the AMS chooses the current model. Since the bound gets tighter as k increases, only one model is eventually maintained on the sensor node. However, since the rationale of the AMS is to maintain a set of models running concurrently on the node, the racing process should be aborted once the cardinality of the set of remaining models reaches a desired size. Furthermore, discarded models could be periodically readmitted into the set to take into account possible changes in the data dynamics. Alternatively, discarded models could be replaced by other candidate models, keeping the total number of maintained models constant.

3.4.3. AMS Algorithm

Algorithms 3.1 and 3.2 show the pseudocode of the AMS and its auxiliary function *updateModel*, respectively. The AMS algorithm takes as inputs the error tolerance e_{max}, the number of candidate models N_{AMS}, the set of models $\{h_n\}$, and their corresponding costs $\{C_{h_n}\}$[9]. The output of the algorithm is the best performing model h^*, to which we also refer to as the *current model*. When data collection starts, the AMS initializes all the models $\{h_n\}$, sets the current model to be the one with the lowest model cost[10], and sends the first update to the sink. Then, at each data collection round k, the AMS runs the function *updateModel* for each model in the set of candidates. The function first updates the model h_n including the newly collected sensor reading X_k and then estimates the new value for the relative update rate $U_{h_n,k}$. To this end, the function computes the current prediction using the *virtual* model $h^v_{n,k-1}$. The parameters of $h^v_{n,k-1}$ are not updated with X_k, since they

[9] The model costs must all be set to 1 if the relative update rate is used as performance indicator.

[10] We recommend to include the constant model in every implementation of the AMS, which has cost = 1, and represents a good default model choice.

represent the parameters that the sink would have used to compute the prediction if h_n were the current model. Thus, a virtual model is only updated when the prediction error exceeds e_{max}, since in this case the node would have accordingly sent an update to the sink. Thus, the function *updateModel* computes the prediction $\hat{X}_k$ using $h^v_{n,k-1}$. Then, depending on the value of the prediction error $\hat{X}_k - X_k$, the relative update rate is computed according to equations 3.11 and 3.12.

After running the function *updateModel* for all models, control returns to AMS, which then behaves as if it were a "classical" DPS scheme. It therefore checks wether the absolute value of the difference between the reading estimation $\hat{X}_k$, computed at the sink using the current model h^*, and the actual sensor value X_k does not exceed e_{max}. If the error threshold is exceeded, the model showing the lowest value for the relative update rate is chosen as the new current model h^*. Accordingly, a model update composed of the current reading X_k, the input values $\mathbf{X}_{h^*,k}$, the model h^*, and the parameters $\boldsymbol{\theta}_{h^*,k}$ is sent to the sink. Then, if required, the racing mechanism is run in order to discard or exchange poorly performing models.

Algorithms 3.1 and 3.2 can easily be adapted to support alternative metrics rather than the relative update rate to determine the performance of the models. For instance, using the weighted relative update rate (cf. section 3.4.1) would require just a few straightforward modifications.

3.5. Experimental Evaluation of the AMS

This section shows the ability of the AMS to ensure that the communication savings obtained using the DPS are always close or equal to those obtainable with the a posteriori best performing model. To perform our experimental study, we implemented the AMS in Matlab using two different sets of candidate models. The first includes the constant model and 5 different autoregressive models (AR-AMS), while the second comprises exponential smoothing models with variable parameters (ES-AMS). We retrieved several data sets from real WSN deployments and used them as our test signals. To be able to compare results from data sets related to different sensors, we also introduced a generalized error threshold k_e. In the following, we first describe the details of our experimental setup, and provide then the actual results in sections 3.5.2 and 3.5.3.

Algorithm 3.1 AMS - Adaptive model selection Algorithm

Require: e_{max}, N_{AMS}, $\{h_n\}$,$\{C_{h_n}\}$
Ensure: $h^* = argmin_{h_n} U_{h_n,k}$
  $k \leftarrow 1$
  $X_k \leftarrow$ getNewReading()
  for $n = 1$ to N_{AMS} **do**
    $h_{n,1} \leftarrow$ initialize(h_n,X_1)
    $h^v_{n,1} \leftarrow h_{n,1}$
    $U_{h_n,1} \leftarrow 1$
  end for
  $h^* \leftarrow argmin_{\{h_n\}} C_{h_n}$
  sendUpdate(X_k,h^*)
  loop
    $k \leftarrow k + 1$
    $X_k \leftarrow$ getNewReading()
    $\hat{X}_k \leftarrow$ predictReading(h^*)
    for $n = 1$ to N_{AMS} **do**
      $h_{n,k}$,$h^v_{n,k}$,$U_{h_n,k} \leftarrow$ updateModel($h_{n,k-1}$,$h^v_{n,k-1}$,X_k)
    end for
    if $(|\hat{X}_k - X_k| \geq e_{max})$ **then**
      $h^* \leftarrow argmin_{\{h_n\}} U_{h_n,k}$
      sendUpdate(X_k,h^*)
      $\{h_n\} \leftarrow$ racing($\{h_n\}$)
    end if
  end loop

Algorithm 3.2 updateModel - Algorithm for model updates

Require: k, X_k, e_{max}, $h_{n,k-1}$, $h^v_{n,k-1}$, $U_{h_n,k-1}$
Ensure: $h_{n,k}$, $U_{h_n,k}$
  $h_{n,k} \leftarrow$ updateModel($h_{n,k-1}$,X_k)
  $\hat{X}_k \leftarrow$ predictReading($h^V_{n,k-1}$)
  $h^V_{n,k} \leftarrow$ updateModel($h^V_{n,k-1}$,$\hat{X}_k$)
  if $(|\hat{X}_k - X_k| \geq e_{max})$ **then**
    $U_{h_n,k} = \frac{(k-1)*U_{h_n,k-1}+1}{k}$
    $h^v_{n,k} = h_{n,k}$
  else
    $U_{h_n,k} = \frac{(k-1)*U_{h_n,k-1}}{k}$
  end if

3.5.1. Experimental Setup

We implemented the DPS using both autoregressive (AR) models [116] and exponential smoothing (ES) models. We provide here some basic notation and notions about these two model classes. We then introduce the data sets used for our evaluation as well as the definition of our generalized error threshold.

Autoregressive Models. We provide an implementation of the AMS using AR models for two main reasons. First, they have been shown to be both theoretically and experimentally good candidates for time series predictions [28, 122]. Second, model parameters can be estimated by the means of the recursive least square (RLS) algorithm [5], which allows adapting the parameters to the underlying time series in an online fashion, without the need of storing large sets of past data. Time series forecasting using AR models is performed by regressing the value X_k of the time series $\mathbf{X}$ at time instant k against the elements of the time series at the previous p time instants $(X_{k-1}, X_{k-2}, ..., X_{k-p})$. The prediction at time $k+1$ is thus obtained as:

$$\hat{X}_{k+1} = \theta_1 X_k + \theta_2 X_{k-1} + ... + \theta_p X_{k-p+1} \tag{3.16}$$

where $(\theta_1, \theta_2, ..., \theta_p)$ are the autoregressive coefficients and p is the *order* of the AR model, which is thus denoted as AR(p). Following the notation introduced in section 3.1.1, let $\boldsymbol{\theta}_{AR(p),k} = (\theta_{1,k}, \theta_{2,k}, ..., \theta_{p,k})$ be the row vector of parameters and $\mathbf{X}_{AR(p),k} = (X_k, X_{k-1}, X_{k-p+1})$ be the row vector of inputs for a model AR(p) at time instant k. Then the scalar product:

$$\hat{X}_{k+1} = \boldsymbol{\theta}_{AR(p),k} \cdot \mathbf{X}^T_{AR(p),k} \tag{3.17}$$

returns the prediction at time instant $k+1$. The parameters $\boldsymbol{\theta}_{AR(p),k}$ can be computed by means of the RLS algorithm, which consists in a computationally thrifty set of equations that allows to recursively update the parameters $\boldsymbol{\theta}_{AR(p),k}$ as new observations become available [5]. The related computational cost for an update of the vector $\boldsymbol{\theta}_{AR(p),k}$ is $3p^3 + 5p^2 + 4p$. For our experimental evaluation, we implemented the AR-AMS using the constant model (CM) and autoregressive models of orders 1 to 5.

Exponential Smoothing Models. *Exponential smoothing* is the technique behind a class of prediction models with excellent predictive ability and very low computational and memory requirements. ES models are currently considered one of the most general and efficient approaches to time series prediction [45, 65, 90]. Furthermore, recently published results cast exponential smoothing methods in a sound theoretical framework showing their equivalence, in terms of predictive ability, to the widely used ARIMA models[11] [45, 64, 65], which are however significantly more expensive in terms of computation and memory usage.

There exist few subclasses of exponential smoothing models, which differ in the number of parameters needed to specify the model and the way these parameters are updated as new data becomes available [65, 90]. The simplest exponential smoothing model computes a weighted average of the past elements of the time series and returns this value as the forecast for the next element. Slightly more sophisticated models include a so-called "trend component" that follows the possible presence of a local linear trend in the time series. Damping parameters or a seasonal component (to take into account non stationarity and periodicity in the data) may also further improve the predictive ability of the model, but often require a disproportional increase in computation and memory usage [65]. A subclass known as *double exponential smoothing* (DES) is widely used in practice and offers a very good trade off between predictive ability and resource consumption [65]. We therefore propose to use this particular subclass of models for implementing the AMS.[12] Using a DES model, the value X_{k+m} of the time series at m time steps ahead of k can be estimated, at time step k, using the following simple linear equation:

$$\hat{X}_k(m) = L_k + m \cdot b_k. \tag{3.18}$$

The values of L_k and b_k can be in turn recursively computed as follows:

$$\begin{aligned} L_k &= L_{k-1} + b_{k-1} + \alpha \cdot e_k \\ b_k &= b_{k-1} + \alpha\beta e_k, \end{aligned} \tag{3.19}$$

where e_k represents the one-step forecast error, formally defined as:

[11] For instance, the simple exponential smoothing model is equivalent to the ARIMA(0,1,1) model; double exponential smoothing (also known as Holt's linear method) is equivalent to the ARIMA(0,2,2) model [122, p. 373].

[12] In particular, we consider here DES models with additive trends. Models with multiplicative trends are less oftenused and rarely provide better performance [65].

$$e_k = X_k - \hat{X}_{k-1}(1) \tag{3.20}$$

The two parameters α and β that appear in equation 3.19, are the *smoothing constants* of the model and may take values in the interval $[0, 1]$. Thus, following the notation introduced in section 3.1.1, a double exponential smoothing model h at time k is described as $h_k = \{X_k, \alpha, \beta, L_k, b_k\}$. Therefore, no past elements of the time series other than the current reading X_k must be stored, and a model update only requires performing few simple operations, as shown in equation 3.19. The forecast $\hat{X}_{k+m}$ is basically a weighted average of past observations, to which recent data contributes with a higher weight than past data. In particular, past readings are weighted with coefficients that decrease exponentially as the time lag from the current reading increases, as a closer inspection of equation 3.19 shows. The time constant of this exponential decrease is determined by the values of the smoothing parameters α and β, hence the name of this class of models. For our experimental evaluation, we considered a set of DES models with parameters α and β varying with steps of 0.1 in the intervals $[0.1, 1]$ and $[0, 1]$, respectively. We point out that the ES model with parameters $(\alpha = 1, \beta = 0)$ corresponds to the constant model.

Data sets. To evaluate the performance of the considered implementations of the AMS, we selected 20 publicly available data sets collected in real WSN deployments. The data sets have been selected so as to represent different test signals in terms of the nature of the observed phenomenon, signal dynamic, sampling frequency, and length. Table 3.2 lists their names and main characteristics.

The *Heater* data measures the temperature of a heater as cold water flows, as reported in [178]. The *I Light* data set relates to the readings collected by the light sensor of mote 7 during the first 11 days of the Intel Lab deployment, which has already been described in section 3.3.3. The *Monte Temp* and *Monte Hum* data sets were collected between March 23, and April 23, 2006, by sensors 3073 and 3074 of node 9, in the Montepaldi Farm deployment [71]. The data sets *Midra ST1*, *Midra ST2* and *Midra ST3* report soil temperature data registered from January 1, 2006 to March 30, 2006, at three different plants[13] cultivated

[13]Midra ST1 refers to plant 1, Midra ST2 to plant 2 and Midra ST3 to plant 3.

in the greenhouse of the Midra[14] Consortium in Florence (Italy). These datasets represent the development of a physical phenomenon within a given time frame but at different sampling locations. The *Monte ST3a*, *Monte ST3b* and *Monte ST3c* data sets report data collected by the same physical sensor but in three subsequent time periods.[15] All data refers to the soil temperature collected by sensor 3073 on node 3 in the Montepaldi Farm deployment [71]. The last 10 data sets have been retrieved from the historical database of the National Data Buoy Center (NDBC) [139], and refer to data collected by buoy 41012 during the whole year 2005. We should point out here that we used the 6 data sets with identifier 5 to 10 in table 3.2 only for the evaluation of the ES-AMS.

Table 3.2.: Data sets used as test signals for evaluating the performance of the AR-AMS and ES-AMS.

$N°$	Data set name	Sensed variable	Sampling	Period	Source
1	S Heater	Temperature	3 seconds	2h30min	[178]
2	I Light	Light	31 seconds	8 days	[19]
3	M Hum	Humidity	10 minutes	30 days	[71]
4	M Temp	Temperature	10 minutes	30 days	[71]
5	Midra ST1	Soil temperature	10 s	3 months	[71]
6	Midra ST2	Soil temperature	10 s	3 months	[71]
7	Midra ST3	Soil temperature	10 s	3 months	[71]
8	Monte ST3a	Soil temperature	$<$ 1min	2 months	[71]
9	Monte ST3b	Soil temperature	$<$ 1min	2 months	[71]
10	Monte ST3c	Soil temperature	$<$ 1min	2 months	[71]
11	NDBC WD	Wind direction	1 hour	1 year	[139]
12	NDBC WSPD	Wind speed	1 hour	1 year	[139]
13	NDBC DPD	Dominant wave period	1 hour	1 year	[139]
14	NDBC AVP	Average wave period	1 hour	1 year	[139]
15	NDBC BAR	Air pressure	1 hour	1 year	[139]
16	NDBC ATMP	Air temperature	1 hour	1 year	[139]
17	NDBC WTMP	Water temperature	1 hour	1 year	[139]
18	NDBC DEWP	Dewpoint temperature	1 hour	1 year	[139]
19	NDBC GST	Gust speed	1 hour	1 year	[139]
20	NDBC WVHT	Wave height	1 hour	1 year	[139]

Generalized Error Threshold k_e. To be able to compare results obtained from different data sets, the error threshold e_{max} is computed propor-

[14]Multidisciplinary Institute for Development, Research and Applications.

[15]Monte ST3a reports data collected from January 1, 2007 to February 28, 2007, Monte ST3b from March 1, 2007 to April 30, 2007 and Monte ST3c from May 1, 2007 to June 30, 2007.

tionally to the range r of the signal, using a given factor k_e. For the data sets described above, we computed the range r as the difference between the maximum and minimum values in the time series. We let the value of k_e vary between a minimum of 0.01 and a maximum of 0.1. The case $k_e = 0.01$ accounts for scenarios in which high precision is required, while $k_e = 0.1$ corresponds to a very rough bound on the tolerated error. For instance, the range of Midra ST1 data set is $r = 31.25°$, and a generic error thresholds k_e of 0.01 corresponds to an accuracy of $e_{max} = k_e r = 0.3^o$. Such an accurate temperature monitoring may be required, for example, for biological studies aimed at the analysis of plant growth [30]. On the other hand, $k_e = 0.1$ corresponds to a tolerance $e_{max} = k_e r = 3°C$, which could be appropriate for a watering system to be triggered.

3.5.2. Performance of the AR-AMS

We discuss now the performance of the AR-AMS, which we implemented using the constant model (CM) and five autoregressive models of orders 1 to 5 (AR1, ..., AR5) as candidate models. We present our results both in terms of relative update rate and weighted relative update rate. We also discuss the convergence rate of the racing mechanism, as well as the average gains (in terms of weighted update rate) obtained as the generalized error threshold k_e increases. For our evaluation, we used both the first 4 data sets listed in table 3.2 and all the NDBC data sets (data sets 11 through 20 in the table).

Table 3.3 reports the relative update rate obtained when running the DPS with model selection based on the AR-AMS using $k_e = 0.01$. Bold faced figures indicate the best performing models, i.e., all models that are not significantly outperformed by the model with the best (i.e., lowest) update rate.[16] As shown in table 3.3, in most cases AR models outperform the CM. It also shows that their performance is usually statistically equivalent, regardless of the model order. However, the CM performed significantly better than any AR model in three time series (namely *I Light*, *NDBC DPD* and *NDBC WSPD*) and yielded similar performance for two time series (*NDBC AWP* and *NDBC GST*). These apparent deficiencies of AR models are due to the nature of those time series, qualitatively characterized by sudden and sharp changes. These abrupt changes cause the variance in the estimation of the AR

[16]Significance is assessed using a one tailed t-test with respect to best model, $p < .05$.

Table 3.3.: Relative update rate for DPS run with the AR-AMS model selection procedure ($k_e = 0.01$). Bold faced numbers indicate models that yielded the best performances (one tailed t-test with respect to best model, $p < .05$).

	CM	AR1	AR2	AR3	AR4	AR5	AMS
S Heater	74	75	61	**59**	**59**	**59**	AR3
I Light	**38**	40	39	40	40	39	CM
M Hum	53	53	**49**	**50**	**49**	**49**	AR4
M Temp	48	48	**45**	**45**	**44**	**44**	AR4
NDBC DPD	**65**	85	80	80	80	80	CM
NDBC AWP	**72**	**73**	**73**	**73**	**73**	**73**	CM
NDBC BAR	51	50	39	39	39	**37**	AR5
NDBC ATMP	39	39	**36**	**36**	**36**	**36**	AR3
NDBC WTMP	27	27	**21**	**21**	**21**	**20**	AR5
NDBC DEWP	57	**52**	**52**	**52**	**52**	**52**	AR3
NDBC WSPD	**74**	84	82	83	83	83	CM
NDBC WD	85	**81**	**81**	**81**	**81**	**81**	AR1
NDBC GST	**80**	81	**80**	**80**	**80**	81	CM
NDBC WVHT	58	**56**	**56**	**56**	**56**	**56**	AR3

coefficients to increase, making the models unstable and thus allowing a simple CM to provide better performances in terms of update rates (with gains of about 15% with respect to AR models for *NDBC DPD* and gains up to 8% for *NDBC WPSD* over a one year period). The last column of Table 3.3 shows the model that yielded the lowest update rate, which was consequently selected by the AMS procedure.

Further, we assess the performances of the AMS in terms of the weighted relative update rate $W_{h_i,k} = U_{h_i,k} * C_{h_i}$ introduced in section 3.4.1. The model costs C_{h_i} are computed assuming that each data sample and parameter can be stored in one byte. Accordingly, the constant model requires 1 byte to be sent to the sink, while the update of an AR(p) model requires $2p$ bytes (p bytes for the initial input values and p bytes for the parameters). The length L_{hf} of both header and footer of a packet, to which we also refer to as the packet overhead, depends on the specific communication protocol. Since the overhead of a TinyOS packet ranges between 12 and 36 bytes, we considered an average packet overhead of 24 bytes for our experiments. Thus, the number of bytes that need to be transmitted in the default monitoring scheme is just $L_{hf} + 1$, while updating a model AR(p) requires sending $L_{hf} + 2p$ bytes. The cost $C_{AR(p)}$ is thus simply computed as the ratio

Table 3.4.: Weighted relative update rate for DPS run with the AR-AMS model selection procedure ($k_e = 0.01$). Bold faced numbers indicate models that yielded the best performances (one tailed t-test with respect to best model, $p < .05$).

	CM	AR1	AR2	AR3	AR4	AR5	AMS
S Heater	74	78	**68**	**70**	76	81	AR2
I Light	**38**	42	44	48	51	53	CM
M Hum	**53**	55	**55**	60	62	66	CM
M Temp	**48**	50	50	54	56	60	CM
NDBC DPD	**65**	89	89	95	102	109	CM
NDBC AWP	**72**	75	81	88	93	99	CM
NDBC BAR	51	52	**44**	47	49	50	AR2
NDBC ATMP	**39**	41	**40**	43	46	49	CM
NDBC WTMP	27	28	**23**	25	27	28	AR2
NDBC DEWP	57	**54**	58	62	67	71	AR1
NDBC WSPD	**74**	87	92	99	106	113	CM
NDBC WD	**85**	**84**	91	98	104	111	AR1
NDBC GST	**80**	84	90	96	103	110	CM
NDBC WVHT	**58**	**58**	63	67	71	76	CM

$24 + 2p/24 + 1$.

Table 3.4 reports the performances of the CM and AR(p) models in terms of the weighted relative update rate, computed according to the considerations reported above. Since the cost of the CM is 1, the first columns of table 3.3 and 3.4 are identical. On the other hand, there is a general deterioration of performances of AR models. Indeed, although AR models still show a better predictive ability, the cost associated with sending their parameters lower their overall performance. Out of all tested time series, AR models only outperformed the CM five times (on *S Heater*, *NDBC BAR*, *NDBC WTMP*, and *NDBC DEWP*). The AR models eventually selected by AMS were AR(2) (three times) and AR(1) (twice). As before, the rightmost column of table 3.4 lists the model that yielded the lowest weighted update rate for each time series.

We further analyze the convergence speed obtained when relying on the racing mechanism. To this end, we considered the first 1000 elements of all the available time series and used the weighted update rate metric $W_{h_i,t}$ to evaluate performance of competing models with a confidence $1 - \delta = 0.95\%$. The number of time steps needed by the racing mechanism to discard poorly performing models depends on the nature of the time series. The convergence to a single best model in less than 1000 time instants was obtained in four cases. For other cases,

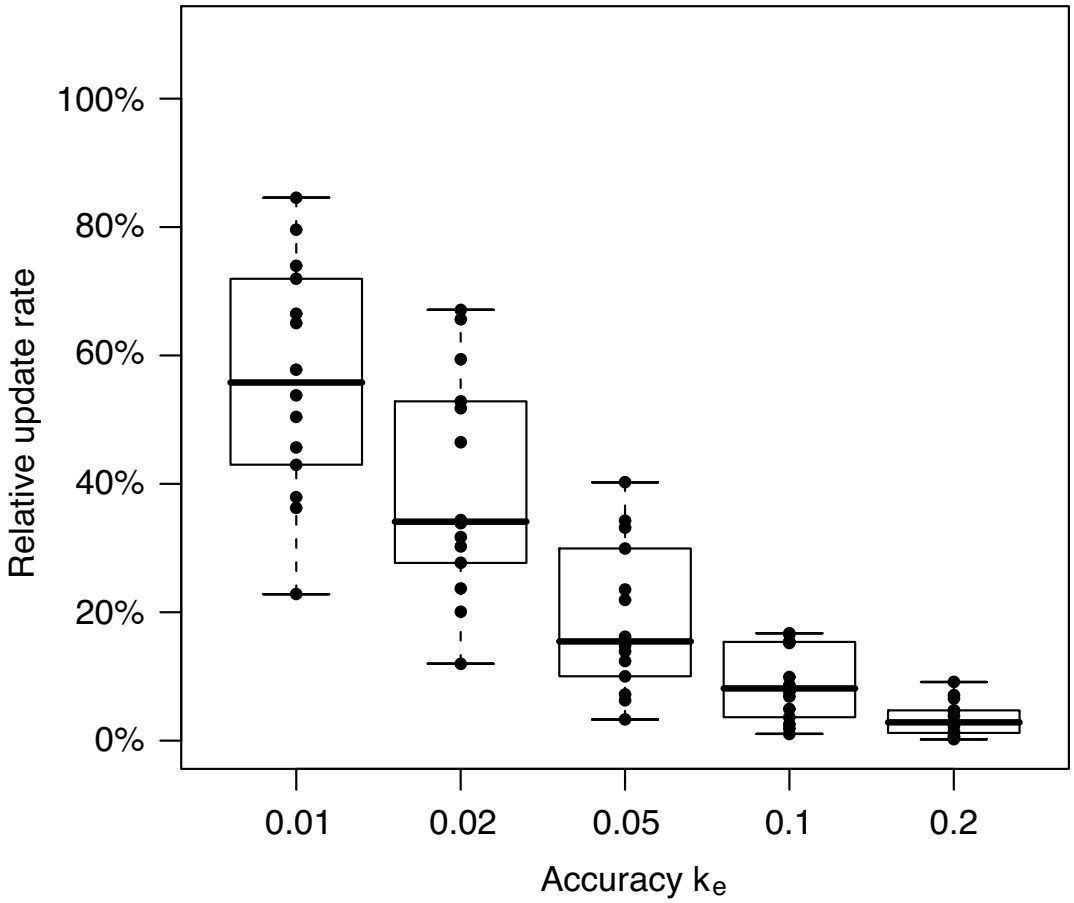


Figure 3.6.: Relative update rate as the generalized error threshold k_e increases.

subsets of two or three remaining models were still in competition after 1000 time instants. The performances of those remaining models were ranging from less than 1% up to 5%, and the a posteriori best model was always part of the remaining set. AR(4) and AR(5) were discarded in all cases due to the overhead incurred in sending their parameters to the sink. For five time series, AR(3) and AR(4) were in the remaining candidates models, while for the other nine time series, either CM, AR(1), or both were still competing after the 1000^{th} time step.

Finally, figure 3.6 reports the relative update rate obtained for each of the considered 14 time series, as the accuracy threshold k_e is relaxed. For these experiments, the AMS was run using the weighted relative update rate of competing models as its performance indicator. Figure 3.6 shows that for $k_e = 0.05$, which corresponds to good approximation of the sensed phenomenon, less than 20% of data actually needs to be sent to the sink (on average) with respect to the default monitoring scheme. The update rate further decreases as the value of k_e increases.

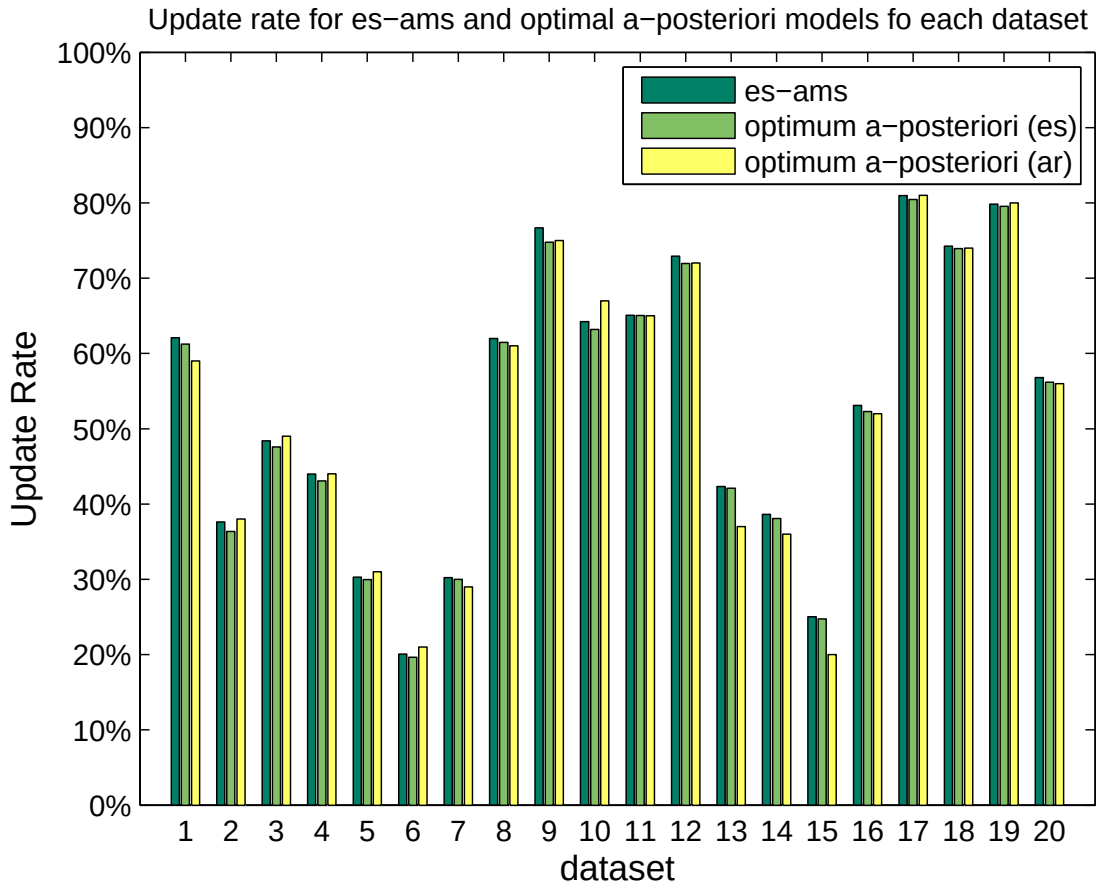


Figure 3.7.: Relative update rates for the ES-AMS and the optimal a posteriori model for each data set.

Furthermore, the predictive capacity of any method tends to converge to that of the constant model as k_e increases. In particular, for values of k_e higher than 0.1, the use of the AR-AMS does not provide, in general, for significantly better gains in terms of update rate than those guaranteed by the use of the simple CM.

3.5.3. Performance of the ES-AMS

As done above for the AR-AMS, we use the relative update rate to assess the performance of the ES-AMS procedure. In particular, for all the data sets reported in table 3.2, we compute, along with the update rate reached by the ES-AMS, both the update rate of the optimal a posteriori model within the exponential smoothing model class, and the update rate of the optimal a posteriori model in the autoregressive model class. The optimal a posteriori model is the model that, among those available, reaches the lowest relative data rate at the end of the

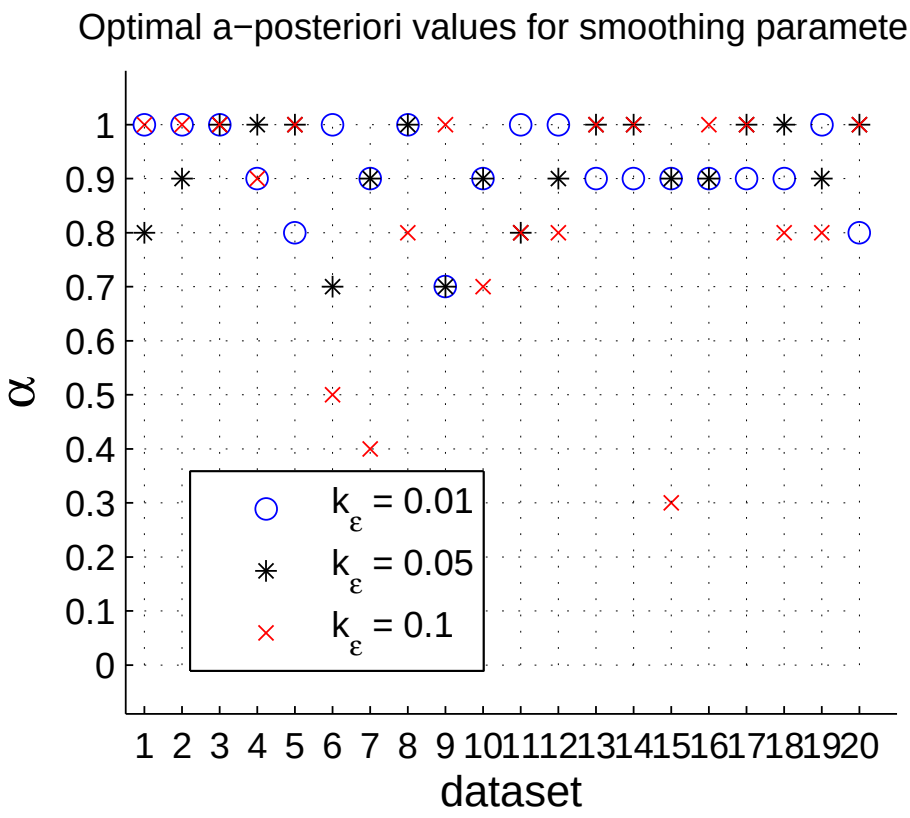


Figure 3.8.: Optimal a posteriori values of the smoothing parameter α for all the 20 considered data sets and three different values of k_e.

observation period. In other words, it is the model we would have liked to know at the beginning of the observation. We computed the optimal a posteriori model also using AR models in order to compare the results obtained with the ES-AMS with those achieved by the AR-AMS. Figure 3.7 shows the relative update rate obtained for all data sets when $k_e = 0.01$. The performance of the ES-AMS in terms of update rate is very close to that of the optimal a posteriori model, for all the 20 data sets. Furthermore, the ES-AMS often outperforms even the optimal a posteriori AR model.

In this chapter, we stated several times that the need for on-line model selection in the context of the DPS is mainly due to the fact that there is in fact no general a priori best model choice. In particular, the predictive ability of a model may depend upon the nature of the data being collected, the default sampling rate, or the approximation threshold e_{max}. Figures 3.8 and 3.9 support this statement by showing the values assumed by the smoothing parameters α and β for the 20 data sets as the generalized error thresholds k_e increases. For instance, for data sets 8, 9, and 10, the value of the parameter α for $k_e = 0.01$

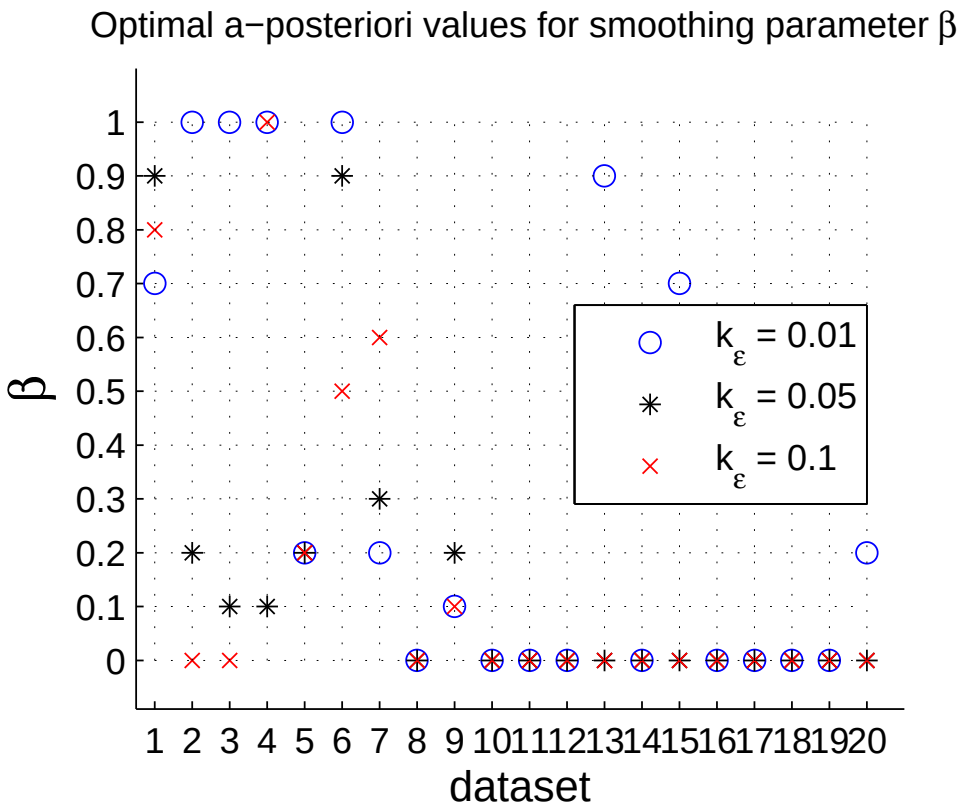


Figure 3.9.: Optimal a posteriori values of the smoothing parameter β for all the 20 considered data sets and three different values of k_e.

is 1, 0.7, and 0.9 respectively. As reported in table 3.2, these data sets correspond to time series captured by one and the same sensor over subsequent time periods. In order to show the effect of a bad model choice of the relative update rate, we consider an example based on data sets 8 and 9. While for data set 8 the best model choice is the constant model ($\alpha = 1, \beta = 0$), the optimal a posteriori model for data set 9 is ($\alpha = 0.7, \beta = 0.1$). For data set 9 the optimal a posteriori model reaches a relative update rate of 76.69%, while the constant model manages to achieve 87.93%. This shows that even for the same sensor, the optimal model choice may vary significantly over time.

3.6. Evaluation of the AMS on a Real WSN Deployment

In this section, we finally report about our experience in testing the ES-AMS framework on a lab-scale WSN deployment. In order to run the AMS on sensor nodes, we implemented it as a TinyOS application, which is described in detail in section 6.2. The application collects

sensor readings at regular time intervals and reports them to a data sink using the DPS strategy. Each time an update is sent to the sink a (possibly new) model is selected from the set of candidates using the AMS strategy. In particular, we implemented the ES-AMS model selection strategy, which constructs the set of candidates using ES models corresponding to different values of the smoothing constants α and β. As we also detail in section 6.2, in our current implementation we let α and β vary, with step 0.1, within the intervals $[0.1, 1]$ and $[0, 1]$, respectively. Thus, the number of models in the set of candidates is $N_{AMS} = N_\alpha \cdot N_\beta = 10 \cdot 11 = 110$, where N_α and N_β represent the considered number of different values of the parameters α and β, respectively.

We thus deployed 10 nodes in our lab and let them run the ES-AMS for several days. The nodes collected temperature readings from the external temperature sensor of the Tmote Sky (see also section 2.2.1). To observe the behavior of the ES-AMS under different operating conditions, we let the nodes collect sensor data at different rates and using different error thresholds. Table 3.5 shows a list of the 10 nodes included in the deployment and their corresponding values of the sampling interval (reported in seconds) and error threshold (reported in degree Celsius). Figure 3.10 shows our experimental setting.[17] The sink node, placed in the lower right corner of the deployment area, forwarded the readings it received from the nodes to a desktop computer running our TinyLAB tool (see section 6.1). Using TinyLAB we were able to immediately import the data in Matlab, visualize it in real-time, and then comfortably perform the offline analysis presented below, which refers to the first two hours of data collection.

To results obtained in this simple experimental setting are qualitatively very similar to those discussed in section 3.5.3. In particular, figure 3.11(a) shows the relative update rate achieved by the ES-AMS for node 1 as a function of time. The red horizontal line in this plot represents the relative update rate achieved by the optimal a posteriori model at the end of the observation period. This line represents the performance eventually achieved by a DPS running a single model, whereby the latter corresponds to the model, within the set of candidates, that achieves the maximal achievable communication savings. As we can see, the performance of the ES-AMS asymptotically con-

[17] The nodes have been connected to a power outlet using USB cables, but they reported data to the sink using wireless communication.

Table 3.5.: Relevant parameter settings for the 10 nodes included in the deployment.

Node ID	**Role**	**Sampling interval (s)**	**Error threshold (°C)**
0	sink	-	-
1	sensor	5	0.1
2	sensor	10	0.1
3	sensor	15	0.1
4	sensor	5	0.5
5	sensor	10	0.5
6	sensor	15	0.5
7	sensor	5	1
8	sensor	10	1
9	sensor	15	1

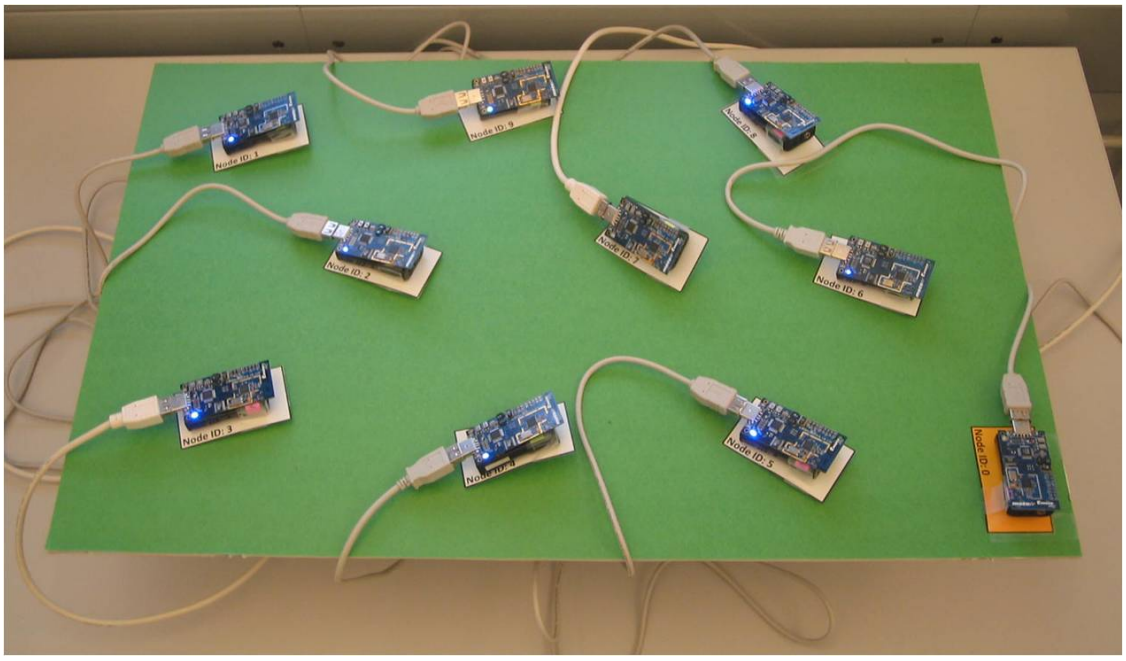


Figure 3.10.: Experimental setting.

verges to that of the a posteriori optimum. Further, figure 3.11(b) and 3.11(c) show the values of the smoothing constants of the current model h^* selected by node 1 as a function of time. The a posteriori optimal model for node 1 has smoothing constants equal to $(\alpha, \beta) = (0.8, 0)$. We can again observe that the ES-AMS eventually selects the "optimal" values of the smoothing constants.

As shown in table 3.5 node 1 represents the sensor operating under the more stringent conditions both in terms of sampling rate and error threshold. To show that the above reported considerations hold also for the other nodes within the deployment, we now consider the performance of node 9, which operates under the less stringent conditions. In particular, figure 3.12 shows, for node 9 the same data as figure 3.11, and allows to make the same considerations reported above concerning the asymptotically optimal behavior of the ES-AMS. However, we should also note that in this case the optimal relative update rate reaches the extremely low value of 0.5%. This is due to the fact that during the observation period the temperature in the room where the nodes were deployed varied mainly within an interval of about $1°C$. Since for node 9 the error threshold was set to $1°C$, only very few updates have been necessary to comply with the defined accuracy requirements. In general, if the variability range of the signal is not at least few times higher than the error threshold e_{max}, the communication gains achievable using the ES-AMS are minimal. In these cases, using a simple DPS running the constant model can usually ensure high performance with minimal effort. In real deployments, however, it is not always possible to know a priori the variability range of the signal, which may anyway vary over time and depend on the specific location of the nodes. For instance, even in our small indoor deployment the variability range of the temperature measured by nodes placed near a window may rapidly sink if the window is left open for about 10 minutes. Such "unpredictable" events may induce an approximatively linear increase or decrease of the signal that the constant model cannot follow efficiently. Therefore, unless a reliable estimation of the signal dynamic is available, the use of the ES-AMS should be preferred over implementations of the DPS that do not provide for automatic model selection. In particular, our results show that the ES-AMS can ensure nearly optimal communication savings without the need of any a priori information on the signals of interest.

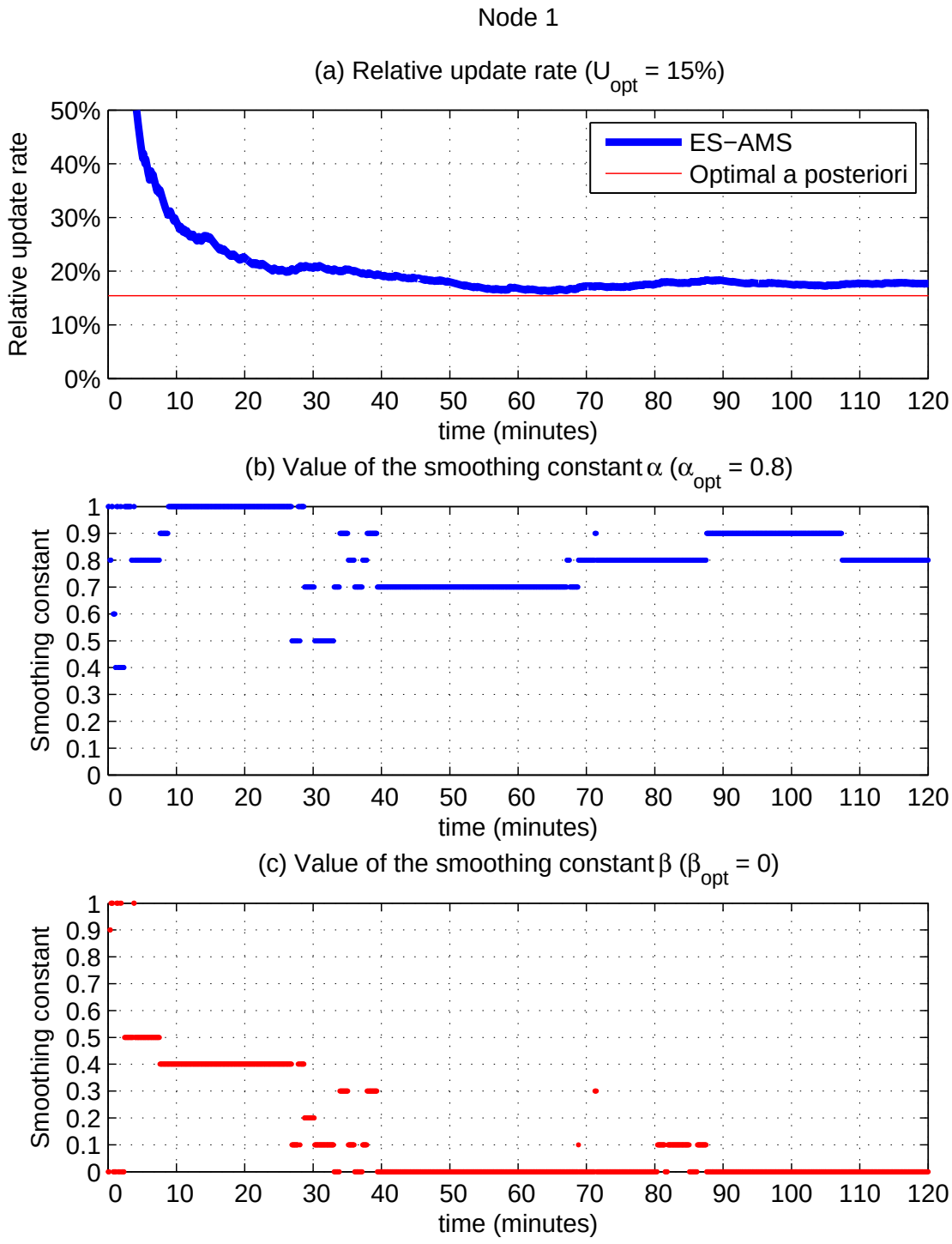


Figure 3.11.: Relative update rate (a), and values of the smoothing constant α (b) and β (b) of the current model h^*, as a function of time. In brackets the correspondent values for the a posteriori optimal model.

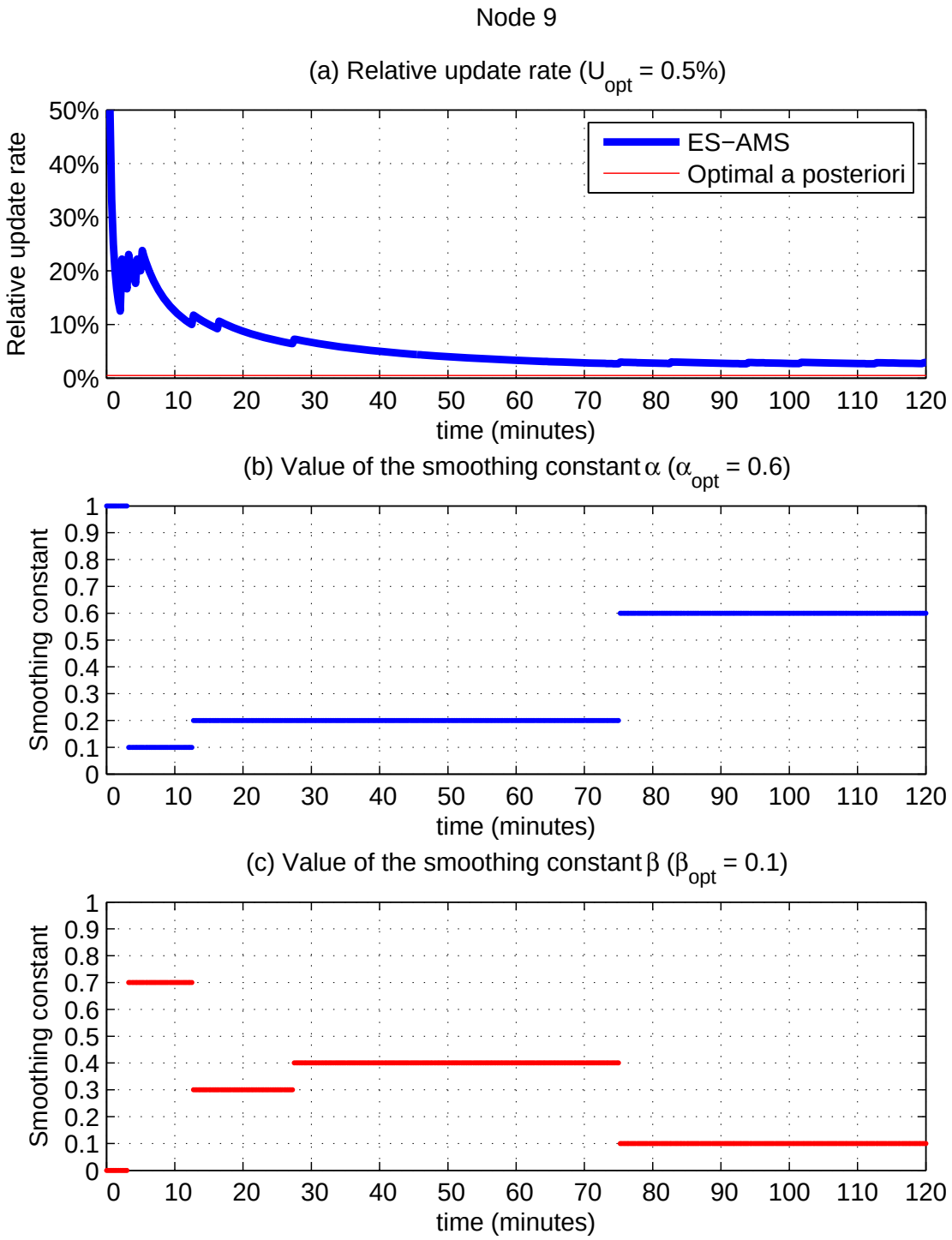


Figure 3.12.: Relative update rate (a), and values of the smoothing constant α (b) and β (b) of the current model h^*, as a function of time. In brackets the correspondent values for the a posteriori optimal model.

3.7. Summary

In this chapter, we focused on the DPS as a generic method to perform temporal sensor selection in WSN. In this context, we first provided an in-depth description of the main characteristics of the DPS and outlined the assumptions it relies upon as well as the requirements it must comply with. Then, we described a lightweight implementation of the DPS based on the LMS adaptive filter. Our LMS-based DPS achieved significant communication savings with respect to the default monitoring mode on several real-world data sets. We then introduced the AMS, an adaptive model selection algorithm that enables sensor nodes to autonomously select, out of a set of candidates, the best performing model to run the DPS. The AMS allows to overcome the main drawback of the LMS-based DPS and several other existing implementations, namely the lack of adequate procedures for automated model selection and online parameter estimation. We suggested two different implementations of the AMS based on autoregressive and exponential smoothing models and analyzed their performance on a large number of data sets retrieved from real WSN deployments. Our experimental evaluation showed the ability of the AMS to provide for a good model choice requiring limited computational and memory resources. Finally, we reported our experiences in running the AMS on a small scale indoor WSN deployment.

4. Spatial Sensor Selection

In the previous chapter, we showed how the temporal data reporting interval of each sensor in a wireless sensor network (WSN) can be controlled using adequate temporal sensor selection strategies. In this context, we assumed that all the nodes in the network collect data at regular time intervals Δ_t, but only a subset of them actually immediately report their readings to the data collector. We believe this approach to be appropriate for small networks and in general when the spatial density of the data is low. But as network size and node redundancy increase, data collected by neighboring nodes may become unnecessarily redundant. Indeed, if the actual density of the deployed nodes is higher than strictly required to comply with the accuracy requirements of the application, making all nodes collect data at each sampling round may no longer be necessary nor convenient. In these cases, spatial sensor selection strategies may come into play to individuate, possibly at each sampling round, an adequate subset of nodes to perform sampling and data reporting.

The selection criteria used to determine the set of active nodes clearly depend on the specific requirements of the application. In this chapter, we focus on applications having the reconstruction of a sensor field as their ultimate goal. We thus investigate the design of spatial sensor selection strategies able to comply with the requirements of such applications. In this context, we first show that the field reconstruction problem can be properly formalized as a coverage problem. We thus suggest to use the well-known coverage configuration protocol (CCP) by Xing et al. [205], as a basic technique to perform spatial sensor selection. We then address some inefficiencies of CCP and propose novel heuristics to improve its performance. In particular, we introduce a technique to rank the relevance of single sensor nodes for the global sensing task. Using this ranking for guiding the sensor selection process, we can significantly reduce the control overhead of CCP. Further, we show that our sensor ranking method can be successfully adopted also to improve the performance of selection strategies based on ran-

dom node activations. Finally, we also consider the possibility to use our sensor ranking strategy to influence the activity of a node as a data router. To this end, we analyze the interplay of our optimized random sensor selection strategy with the CTP data collection protocol [58,68].

In the next section 4.1, we provide a more detailed definition of the field reconstruction problem in WSN. We then review related work in section 4.2. In section 4.3, we show under which assumptions the field reconstruction problem can be reduced to a coverage problem. We then summarize and discuss the relevant aspects of the CCP protocol, as well as its applicability to the field reconstruction scenario in section 4.4. We then introduce our sensor ranking technique and optimized version of CCP in section 4.5. In section 4.6, we report experimental results showing the ability of our sensor ranking strategy to reduce the overhead of CCP. In section 4.7, we present and evaluate the performance of our sensor ranking strategy when used in conjunction with a random sensor selection scheme. We further discuss the interplay between sensor selection and routing in section 4.8. Finally, section 4.9 summarizes and concludes the chapter.

4.1. Field Reconstruction in WSNs

In a typical monitoring application based on WSNs, the goal of the network consists in capturing, possibly at regular time intervals, the values of a sensor field over a target area. Since a WSN can only sample the sensor field at discrete, typically irregularly spaced locations, adequate reconstruction algorithms to compute the values of the field at any point of the region of interest (RoI) must be applied. Leveraging the terminology commonly used in image processing literature [75,132,180], we refer to the process of computing the values of the sensor field over the target area, starting from the samples collected by the WSN, as the *field reconstruction* or *field recovery* process. Further, we refer to the set of positions at which the samples are collected as a *sampling geometry* or *sampling pattern*. If the samples are collected over a regular grid we accordingly speak of *regular* or *uniform* sampling geometry [132]. Since the positions of the samples coincide with the positions of the nodes that collected them, a subset of nodes actively sampling the sensor field constitutes a sampling geometry. In this context, sensor selection strategies can help in individuating adequate subset of nodes, and thus sampling geometries, whose readings, once reported at the

central data collector, can enable reliable reconstruction of the sensor field of interest. To this end, the characteristics and requirements of the specific algorithm used at the central server must be known. In the following, we briefly discuss the criteria that, in our opinion, should guide the choice of an adequate reconstruction algorithm to be used in the context of WSNs.

The field reconstruction problem has been studied in several research fields beyond WSNs, like in computer vision, and for medicine, astronomy, or geophysics applications [126, 146]. Accordingly, a vast literature on theoretical principles and practical algorithms for performing field reconstruction is available [126, 195]. Several techniques, however, cannot be applied to reconstruct sensor fields sampled by a WSN. Indeed, many algorithms require the samples to be available over a grid [146, 171, 195], while others work properly only for sampling geometries resulting from the perturbation (jittering) of a uniform pattern [42, 126]. Clearly, these assumption are not likely to be met for the sampling patterns typically offered by a WSN.

Indeed, in real WSN deployments the actual spatial distributions of the nodes may be highly irregular and poorly controllable [10, 19, 37, 185, 186, 190]. Irregularities may result from specific characteristics of the terrain and practical difficulties in placing the nodes with care at the "correct" positions [63]. Furthermore, nodes could move after deployment due to the action of weather (e.g., wind, rain), animals or humans. Therefore, assuming a regular, or controlled, placement of sensor nodes in WSNs is often unrealistic. For this reason, algorithms performing reconstruction from samples collected by a WSN must be able to cope with arbitrary, irregular sampling geometries.

In this context, we found the ACT reconstruction algorithm [56, 75, 169, 180] to be particularly suited to be used in WSN settings. The ACT is a well-known technique to process medical or geophysics data [158, 182] and is particularly robust against the presence of large gaps between, or dense clusters of, samples [75]. Furthermore, it has been shown to achieve better reconstruction performance with respect to other specialized algorithms and with different kind of data [75, 158, 182]. Although the ACT is a well-known, generic technique to perform reconstruction from scattered samples, it has received only little attention in the WSN literature [143, 152]. In the context of our work, we assume the ACT to be the reconstruction algorithm of choice. With this assumption, justified by the wide applicability and good performance

of the ACT, we show that the problem of selecting favorable sampling geometries for field reconstruction can be reduced to a coverage problem. This makes it possible to leverage coverage preserving algorithms as sensor selection strategies in the field reconstruction scenarios we are considering. In particular, the well-known coverage configuration protocol (CCP) [205], can be adapted without difficulties to our application context, although it was tailored for surveillance and target detection applications.

Before going into further details, however, we first review related wok in the following section 4.2. We will then come back to the ACT algorithm and the requirements it poses on the sampling geometry in section 4.3.

4.2. Related Work

Several authors within and beyond the WSN research community investigated the spatial sensor selection problem, contributing a large number of interesting approaches. In the following, we introduce the contributions that most closely relate to our own work and outline their main merits as well as possible drawbacks. For the sake of simplicity, we classify related work into six different categories and discuss it in just as many subsections. However, some efforts may belong to and thus be mentioned within more than one category.

We start our exposition with the presentation of approaches focusing on the *field reconstruction* problem in WSNs. Since we reduce the problem of sensor selection for the purpose of field reconstruction to a coverage problem, we move on presenting relevant *coverage preserving* algorithms. Afterwards, we review approaches leveraging *random sampling* techniques to perform spatial sensor selection. These latter approaches are relevant to our work since they relate to the contributions presented in section 4.7. For the sake of completeness, we also briefly review *utility-* and *model-based* sensor selection algorithms as well as techniques focusing on the *computation of aggregates*. The approaches belonging to these three latter categories, however, only partially relate to our work. Therefore, skipping their discussion will not hamper the reader to follow the rest of the chapter.

4.2.1. Field Reconstruction

Sensor selection algorithms focusing on field reconstruction applications aim at controlling the number and spatial distribution of the nodes so as to enable an accurate recovery of the sensor field. Several authors addressed the thereby arising challenges and proposed interesting approaches to deal with them on both the theoretical and practical side [16, 37, 94, 95, 123, 127, 140, 143, 152, 155, 157, 175, 203, 213].

Willet et al. reckon that "*high spatial densities of sensors are desirable for achieving high resolution and accurate estimates of the environmental conditions, but high densities also place heavy demands on bandwidth and energy consumption for communication.*" [203]. To help reduce the number of sensing nodes in such high density scenarios, they propose a two-step approach based on a *preview* and a *refinement* step. The network is first divided into regular cells using recursive dyadic partitioning. Each cell corresponds to a logical cluster, and for each cluster a node is assumed to take over the role of clusterhead. In the preview phase, a subset of the nodes samples and reports data to the sink. As data makes its way towards the sink, each clusterhead performs data aggregation by fitting piecewise linear models called *platelets* [204] to the sensor measurements. The more homogenous the data, the fewer number of platelets (and, thus, number of bits) is needed to represent them. Since the sensor field is assumed to be piecewise homogeneous, the use of platelets allows to significantly reduce the amount of information that must eventually reach the sink. Across the boundaries between homogenous regions, however, the field exhibits abrupt spatial changes and thus the possibility to aggregate data decreases. Observing the aggregated data, the sink can individuate (boundary) regions with high information content and thus trigger a refinement step to gather additional data from these regions. This approach, dubbed *Backcasting*, has proven very efficient in detecting boundary regions and thus allowing to reconstruct piecewise smooth fields. However, the approach does not extend to the general field reconstruction setting we consider in our work. Additionally, the sampling pattern is considered to be a regular grid and the extension of the approach to irregular geometries is not investigated.

Other approaches elaborating on the classical rate-distortion problem, like those reported in [123, 155, 156], assume uniform sampling geometry and sensor fields with very specific characteristics (e.g., sta-

tionary Gaussian fields).

Moving beyond the assumption of uniform nodes deployments, the approach presented in [152] lets the sensor nodes construct a sampling geometry that resembles a binary *blue noise* sampling pattern. The main feature of a blue noise pattern consists of having a spectrum with very little low-frequency content and no concentrated spikes of energy [132]. Thanks to these characteristics, fields sampled using a blue noise pattern can be reconstructed without aliasing effects. Or, better said, aliasing appears on the recovered field as diffused noise instead of in the form of visible artifacts [42, 132]. There exist standard methods to generate a blue noise pattern starting from some random distributions, and the authors of [152] present a modification of such methods that is suitable to be used in WSN settings. A distributed algorithm to generate approximate blue-noise sampling patterns is described, while performance is evaluated considering an optimal, centralized solution. The main rationale of the distributed algorithm is to make nodes decide about their deactivation by setting appropriate backoff timers. Nodes whose deactivation timers exceeds some pre-defined threshold will remain active and contribute to the generation of the blue noise sampling geometry. The main disadvantage of this approach lies in the need to transmit the deactivation beacons, especially considering that the algorithm tries to maximize the number of nodes that deactivate themselves.

Dong et al. [48] investigate the impact of the sampling geometry on the quality of the reconstruction of a 1-dimensional signal. They consider both uniform and random sampling geometries and resort to a nearest neighbor linear estimator that minimizes the MMSE[1] as a reconstruction technique. Their asymptotic analysis shows that, in the case of high signal-to-noise-ratio (SNR), uniform geometries allow to achieve significantly better reconstruction with respect to random ones. However, the gains in selecting a uniform sampling pattern shrink as the SNR decreases. This latter result is particularly interesting since it suggests that, under the given assumptions, random sensor selection can provide for performance comparable to that of more complex selection schemes. We elaborate more on the potential of random sampling in sections 4.2.3 and 4.7.

In a series of publications Nordio et al. [140–143] studied some theoretical issues related to the problem of field reconstruction from nonuni-

[1] Minimum mean square error.

form samples in the context of wireless sensor networks. In particular, they investigate the performance of linear reconstruction filters for bandlimited signals [142]. They provide analytical expressions for the mean square error (MSE) of the reconstruction in the asymptotical case in which both the bandwidth of the signal and the number of nodes grows to infinity [141, 142]. In their study, the authors also investigate the influence of noise in the measurements and errors in the estimation of the positions of the nodes [142]. Further, they show that in order to analyze the reconstruction problem exactly it is necessary to dispose of the analytical expression of the eigenvalue distribution of the reconstruction matrix [143]. Since this expression is unknown, they provide an approximation thereof and use it as the basis for their analysis. The results presented by Nordio et al. [140–143], although retrieved for an asymptotical case and mostly only for 1-dimensional fields, allow to characterize the reconstruction performance of specific linear filters with respect to several parameters like the number of sensing nodes or the level of noise in the data. In our approach, we do not consider theoretical performance in terms of MSE but focus on more practical methods for providing appropriate sampling geometries.

4.2.2. Coverage Preserving Algorithms

In surveillance and target detection and tracking scenarios, WSNs are typically required to provide spatial coverage over a region of interest at each time instant. In these scenarios, sensor nodes are equipped with sensors enabling the detection of the phenomenon of interest, e.g., the presence of a vehicle or person, with good accuracy up to a certain distance, which is referred to as the *sensing range* R_s of the node. For instance, typical infrared sensors may allow to detect a human intruder present at up to $R_s = 3m$ from the node itself. Assuming isotropic sensor behavior, the area covered by a node can be modeled as a discus $D_{R_s}(c)$ having the node itself as its center c and a radius given by the sensing range R_s. Guaranteeing constant coverage thus requires scheduling node activations so that, at each time instant, each point of the RoI lies within the sensing range of at least one node. More precisely, this type of coverage is known in the literature as *1-coverage*. Generalizing the definition, a sensor selection algorithm can guarantee *k-coverage* if, at each time instant, each point of the RoI is within the sensing range of at least k sensors [205, 211]. For instance, points A, B,

and C in Figure 4.1 are, respectively, 1-,2-, and 3-covered, while point D lies outside the area covered by the nodes and is therefore *uncovered* or 0-covered.

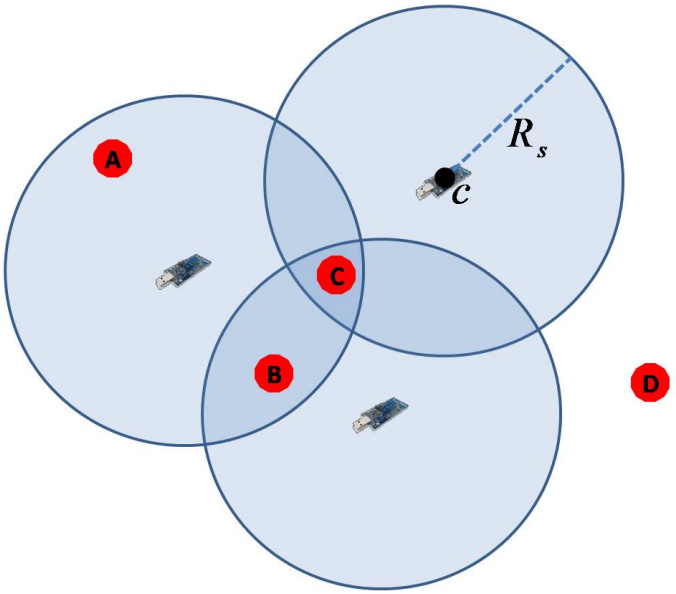


Figure 4.1.: Area covered by three nodes having all the same sensing range R_s. Points A, B, C, and D are 1-, 2-, 3-, and 0-covered, respectively.

Coverage preserving algorithms represent an important category of spatial sensor selection methods, and their use is relevant also for field reconstruction applications. Indeed, as we mentioned in the introduction to this chapter and in section 4.1, the problem of providing favorable sampling geometries for the ACT reconstruction algorithm may be reduced to the problem of finding a set of nodes that guarantees for 1-coverage of the RoI.

A central contribution in analyzing the coverage problem in WSNs is offered by Xing et al. in [205]. The authors present theoretical results relating the two concepts of connectivity and coverage. In particular, they show that if the radio range of the nodes R_{tx} fulfills the condition $2R_s \leq R_{tx}$ and the RoI is 1-covered in the sense we explained above, then 1-connectivity is also guaranteed. The result is also extended to the general case of k-coverage and k-connectivity[2]. Besides these important theoretical results, the authors also present a coverage configuration protocol, dubbed CCP, which we describe in detail in section 4.4. Nodes running CCP decide upon their activation by evaluating if their sensing area is already covered by other nodes that previously declared themselves as active. If a node finds itself to be covered by active

[2] A network is said to be k-connected if, to disconnect it, k nodes must be removed.

neighbors, it remains idle. Otherwise, it becomes active and communicates its decision through a dedicated broadcast message. Possibly, a node can decide to withdraw from the set of active nodes, if it is eligible to do so. In sections 4.4 and 4.5 we show how it is possible to modify the CCP so as to reduce its overhead and thus make it better suitable to be used in the field reconstruction scenario we are considering.

Other coverage-preserving algorithms work along the same rationale of the CCP protocol [188]. However, they propose a different technique to determine whether the sensing area of a node is covered by its active neighbors. Instead, the PEAS algorithm presented in [209] uses a different, pull-based approach. To determine whether to become active or not, sensor nodes broadcast a *probing message* appropriately setting their transmission range to the desired probing range R_p. Active nodes receiving such probing messages also broadcast a reply to signal their activity. If the probing node does not receive any of these replies before a timeout expires, it becomes active. Otherwise, it turns itself off until the next probing round. This approach works well under the assumption of isotropic antenna patterns and adjustable transmission ranges. Unfortunately, both assumptions are hardly met in real WSN deployments.

Several authors also considered the coverage problem in settings in which the active nodes are selected (or deployed) at random over the RoI [40, 110, 111, 200]. For instance, the results presented in [110], recently reprinted in [111], show the asymptotic conditions necessary to guarantee that a region is (almost always) k-covered by a set of n nodes. The nodes are assumed to have all the same sensing range r and be active or inactive with probability p and $(1 - p)$, respectively. The results are reported for the case in which the RoI is the unit square and the nodes are deployed on a uniform grid, uniformly at random or according to a Poisson distribution with rate n. The conditions for k-coverage, although derived for the asymptotic case in which the number of nodes n goes to infinity, can be used to compute rough estimates also for real WSN deployments, as claimed in [110, 111]. In [200] Wan and Yi consider the same problem for both the cases in which the nodes are deployed according to a Poisson or uniform point process. In particular, they show how the probability of coverage changes as the sensing range and number of nodes vary. Furthermore, they note that their results and those reported in [110, 111] are not consistent, probably due to a different handling of the boundary conditions. For

further results we refer the interested reader to [40, 66, 79] and [210]. In section 4.7, we show how these theoretical results can influence the design of sensor selection strategies based on random node activations. Interesting approaches based on random sampling are also discussed in the following section 4.2.3.

4.2.3. Random Sampling

Random sampling occurs in WSNs when sensor nodes decide about their participation in, or are selected for, sensing using a randomized approach. For instance, in a random sensor selection (RSS) scheme, sensor nodes are active with probability p and idle with probability $1 - p$. Assuming all nodes share the same value of p, the expected total number of active nodes at each data collection round is $p \cdot N_{tot}$, where N_{tot} is the total number of nodes in the network. If data packet losses are rare, $p \cdot N_{tot}$ also approximates the average number of sensor readings reaching the sink. The appeal on RSS as a sensor selection strategy clearly lies in its simplicity and very limited control overhead. Furthermore, it provides for a straightforward way to balance the (sensing) workload across the nodes. Furthermore, in some cases RSS may perform comparably to, or even better than, other, more sophisticated sensor selection strategies [120].

In field reconstruction applications, however, the simple RSS strategy may also incur significant performance losses with respect to other approaches [48, 152]. In particular, the blue noise sampling strategy [152] described in subsection 4.2.1, clearly outperforms the plain RSS. Dong et al. [48] also investigate the performance of the RSS with respect to a scheduling approach that activates a deterministic set of equally spaced nodes. For a 1-dimensional field, they provide asymptotical expressions of the reconstruction error (distortion) with respect to the number of collected samples. Their analysis shows that in scenarios in which the signal to noise ratio (SNR) is high, the advantage of constructing uniform sampling geometries is substantial. When the SNR is low, however, the performance loss due to the use of the RSS is drastically reduced. Furthermore, they confirm previously presented results showing that in the case of nonuniform sampling the accuracy of the reconstruction is mainly determined by the maximum distance between any two adjacent samples [56, 181]. Furthermore, they also provide an analytical expression for the probability distribution of the maximum

distance between samples.

We should notice that the above mentioned efforts always considered the simple RSS approach in which the probability of activation p is fixed and equal for all the nodes. As we show in section 4.7, however, making this value depend upon local information available at the nodes may significantly improve the performance of a RSS strategy.

In [96] Iyer and Kleinrock present an interesting approach that eventually makes sensor nodes individually tune their probability of activation p. In particular, they model each node as a player of the Gur Game [193]. In this game a player repeatedly and autonomously takes a binary decision (i.e., "yes" or "no") and communicates it to a "referee". The referee then calculates a function $r(k)$ of the total number of players k that took a positive decision. Each player receives this value as a feedback on her previous decision, which is evaluated to have been "correct" with probability $r(k)$ or "wrong" with probability $1 - r(k)$. Each player can thus take her next decision on the basis of the feedback from the referee. A high value of $r(k)$ signals that the referee did not approve the global behavior of the players and thus wants to force them to modify their decisions. In a WSN scenario, the central server takes over the role of the referee and sends its feedback to the nodes after each data collection round. Using the Gur Game approach, the server can induce the nodes to modify their individual behavior in order to reach a global, common goal, specified as achieving the optimal number of active nodes. The question of whether this approach can actually bring the network to a stable, optimal state, remains unanswered. Furthermore, providing feedback to all nodes in the network after each data collection round may require a disproportionate amount of communication. For the sake of completeness we should also mention that this approach is only apparently a random sampling scheme. Indeed, a node is always either in a "stay idle" or "participate" state, but the transitions between these two states depend on the probability $r(k)$ and a pre-specified inertia.

Other approaches investigating random sensor selection techniques include [14]. In their work Bash et al. [14] focus on applications requiring the computation of aggregates from samples picked uniformly at random over the set of all nodes. In these scenarios, all sensors are required to report their data with the same probability p, irrespective of their position and local density. The authors propose an approximate solution to this problem that select specific locations in the networks to

be sampled and then uses geographic routing to route the query to the node s closest to this location. The node then decides whether to accept the task and report its data, depending on both the area of its Voronoi cell $A(s)$, which must be precomputed, and a user-defined threshold τ which is distributed to all nodes. In particular, the node reports its value with probability $min(A(s), \tau)/A(s)$. This simple heuristic manages to balance the probability of nodes with high and small Voronoi cells to report their values. This method differs from our approach since it aims at "flattening" the areas covered by the nodes, while we aim at exploiting the differences in these values to optimize the participation of the sensors in sensing and communication. Furthermore, the method proposed in [14] doesn't scale well with the number of samples needed and the size of the network, since it may require several attempts to retrieve one single sample, thereby possibly sending multiple requests along very similar paths.

4.2.4. Utility-Based Sensor Selection

In section 1.1 we introduced the idea that the sensor selection problem can be treated formally as an optimization problem in which the cost function summarizes the energy expenditures of the network while the utility function captures the information content of the data. Several authors addressed the problem along this line and provided interesting contributions [23, 33, 34, 44, 98].

In [33] and [34], for instance, Byers and Nasser provide a simple framework to perform utility-based sensor selection in WSNs, also including a combined optimization of routing and sensing roles. They, however, focus on the general applicability of their approach and the heuristic used to make sensors decide upon their role does not include any considerations related to the actual accuracy of the sensed data. In particular, the authors underline "*the importance of providing geographically distributed sets of reporting sensors*" for many applications, but do not focus on this issue in their work. As we will detail in section 4.3, the actual spatial distribution of the sensing nodes is crucial in determining the accuracy of the reconstruction of a sensor field.

In their work Byers and Nasser considered a global utility function whose value monotonically increases with the number of sensing nodes. They also underline that in many practical scenarios there will be a *diminishing marginal return*, i.e., the advantage of adding new data

diminishes as the number of total collected samples increases. Bian et al. studied the formal properties of this class of utility functions, also dubbed *submodular* functions, in the context of WSNs [23]. In particular, they show that the utility-based sensor selection problem can be expressed as a linear program of polynomial size and can thus be solved exactly in polynomial time. Computing such solution, however, requires all sensor utilities and costs to be known at a central location, which is clearly unpractical in typical WSN settings. This result is, however, particularly interesting since it allows to set a benchmark by indicating the optimal, centralized solution towards which the performance of other, distributed approaches can be evaluated. In the same work Bian et. al also studied the properties of supermodular utility functions, for which "*the benefit of combining two (disjoint) sets* [of sensors] *is at least as large as the sum of the individual benefits*" [23]. For this class of utility functions, however, they conjecture the sensor selection problem to be NP-complete and show other related theoretical results.

Focusing on the computation of aggregates like mean, median and maximum, Das and Kempe [44] show that the solution of the sensor selection problem can be found solving an appropriately defined k-median problem, for which good and practical approximation algorithms are known. Other approaches propose elegant theoretical frameworks to compute approximate solutions to the sensor selection problem [98] but they rely on centralized computations and the proposed results are thus hardly applicable in real WSN settings. In our approach to the sensor selection problem, we do not resort to the utility-based formalization discussed above. Instead, we rely on existing practical protocols and aim at improving their performance.

4.2.5. Model-Based Sensor Selection

Model-driven sensor selection strategies have been widely investigated to optimize data collection in WSNs [46, 76, 144, 198, 206, 212], as also already discussed in chapter 3. In our work, we avoid the use of predefined models to describe the signal of interest and make use of only limited a-priori knowledge. This allows to design more generically applicable and robust sensor selection strategies. However, if the signal dynamics are stable and known in advance, model-based approaches represent valuable and interesting alternatives.

For instance, knowing that the sensor field of interest can be well represented in a certain function space enables the adoption of techniques seeking for the computation of the coefficients of the signal in this basis instead of a complete data collection [76]. Clearly, this allows to save communication, since in place of a large amount of raw data, only few coefficients must be sent to the data collector. On the other hand, determining the proper function space is not trivial, but crucial for the successful adoption of this technique. Desphande et al. [46] suggest to build a multivariate model of the data collected by the network and use it to answer user queries. To build such model, they exploit spatio-temporal correlations in the data, whose occurrence must however be built-in a-priori. If the uncertainty on the query answer is higher than a given threshold, the model is updated collecting new data from selected sensor nodes. If analytical models of the sensor field and its correlation structure are known, a large set of distributed source coding techniques also becomes available [155, 156, 170, 206]. However, as noted in [81], "*in many applications prior knowledge of the precise correlation in the data is unavailable, making it difficult or impossible to apply such distributed source coding techniques.*"

4.2.6. Computation of Aggregates

Sensor selection algorithms also come into play when the network is required to compute aggregate values over the RoI instead of providing complete field reconstruction. Aggregate information like mean, median or maximum value of a sensor field or network parameter may indeed be of great interested for many applications. Although our work considers the problem of complete reconstruction of a sensor field, and not on the computation of its aggregates, we briefly mention some interesting approaches the reader is referred to.

In [120], Lin et al. propose *region sampling*, a technique to compute approximate aggregates when only a pre-specified energy budget is available. Region sampling partitions the network in k regions, within which sensor values are collected and aggregated. With an adequate choice of the regions, the approximation error can be bounded. To this end, a set of statistics is collected during network operation and used to perform a centralized choice of the nodes from which samples must be retrieved. As already discussed in section 4.2.3, Bash et al. [14] propose a method for enabling computation of aggregates for scenarios in which

all sensors are required to report their data with the same probability p. In [109], Kuhn et al. provides theoretical results to address the k-selection problem, consisting in finding, out of a set of n element, the k^{th} smallest of these elements. They show that this problem can be efficiently solved using both randomized and deterministic algorithms. Das and Kempe [44] propose a technique to select nearly optimal subsets of sensors that can predict the value of certain aggregate functions within a given error.

4.3. Irregular Sampling in WSNs and the ACT Reconstruction Algorithm

The problem of reconstructing a sensor field from its irregular samples has been studied in several different contexts, and has received increasing attention in the last decades [126]. Thanks to the achieved results, the mathematical theory of irregular sampling is by now well-established. However, practical solutions to perform reconstruction from scattered samples are still scarce. One of the major problems in this context is the definition of the formal conditions that the sampling geometry must fulfill in order to make the reconstruction problem numerically tractable [126]. In a series of publications Feichtinger, Gröchenig, Strohmer and Scherzer [56, 75, 169, 180] address this problem and provide a robust and efficient numerical method for field reconstruction from irregular samples. Their algorithm, known as the ACT, draws upon the observation that fitting the samples $f(s_i)$ of a spatial field by a trigonometric polynomial[3] p (of appropriate order and period) makes the reconstruction problem numerically tractable. Furthermore, they provide extensive experimental results showing the superior reconstruction performance and higher computational efficiency of their method with respect to other approaches. Last but not least, they also provide formal requirements the sampling geometry must fulfill in order to enable robust and efficient reconstruction.

The ACT has often been used to process data in medicine or exploration geophysics [158,182], but has received only little attention in the WSN literature [143, 152]. In the context of our work, we suggest to resort to the ACT algorithm [56, 75, 169, 180] to perform sensor field

[3] A trigonometric polynomial is a finite linear combination of sine and cosine functions. The order M of the polynomial indicates the number of different frequencies for which sine and cosine functions are generated.

reconstruction in WSNs. Its use allows to achieve good reconstructions without posing unrealistic requirements on the sampling geometry. Furthermore, experimental results showed its ability to cope with datasets of different nature and size [158, 182]. Therefore, the ACT appears a suitable tool for performing reconstruction from samples collected in WSN settings. For the interested reader, we provide a detailed description of the ACT in appendix A. In the following, we briefly summarize its main characteristics and focus on the requirements the sampling geometry must fulfill to enable robust and efficient reconstruction. In particular, we show how the use of the ACT allows to reduce the problem of selecting favorable sampling geometries to a coverage problem.

We first consider a 1-dimensional sensor field f sampled at r scattered locations $s_j, j = 1, \ldots, r$ over the segment $[0, 1]$. Without any loss of generality, we assume the sampling points to be numbered so that $0 \leq s_1 < s_2 < \ldots < s_r < 1$. The ACT reconstructs the sensor field f by fitting its samples with a trigonometric polynomial p_M of order M and period 1. The optimal reconstructing polynomial p_M^* is the one that solves the least squares problem:

$$\sum_{j=1}^{r} w_j |p_M^*(s_j) - \widetilde{f}(s_j)|^2 = minimum\, in\, \mathcal{P}_M, \tag{4.1}$$

where the minimum is taken over the space of all polynomials $\mathcal{P}_M$ of order M and period 1. The order M of the polynomial is determined by the bandwidth B_s of the field f, while the weights $w_j, j = 1, \ldots, r$ depend on the sampling locations only. We show in appendix A how to properly set these values. If the bandwidth of the signal is unknown, the so-called multilevel version of the ACT, dubbed ML-ACT, allows to estimate it on the fly, as reconstruction is performed [169]. The set of $2M + 1$ coefficients a_M^* that generates the polynomial p_M^*, is the solution to the linear system:

$$a = T^{-1} b. \tag{4.2}$$

T is a square matrix of dimensions $(2M + 1) \times (2M + 1)$ and b a vector of length $2M + 1$. As shown in appendix A, the matrix T has a Toeplitz structure and, thus, efficient methods to perform its inversion, necessary for solving system 4.2, become available. In their work, Feichtinger at al. suggest to use the conjugate gradient iterative method [70] to perform the inversion of T. The name ACT actually

summarizes the main features of this algorithm: the presence of the adaptive (A) weights w_j, the use of the conjugate (C) gradient method, and the Toeplitz (T) structure of the matrix T.

For a (unique) solution of system 4.2 to exist, the invertibility of T is a necessary and sufficient condition. The actual quality of the reconstruction, however, also depends on the spectral properties of the matrix T, which are characterized by the value of its condition number[4]. In particular, the lower the condition number, the more robustly and efficiently the solution of system 4.2 can be computed. Thus, the possibility to bound the value of the condition number $k(T)$ of T, allows to formally characterize the reconstruction performance of the ACT. Since the entries of the matrix T depend on the sampling geometry only, controlling the values of the sampling locations s_j allows to control the performance of the ACT[5].

In the 1-dimensional case, it can be shown that the matrix T is invertible if at least $2M + 1$ samples are available. Furthermore, its condition number can be guaranteed to be bound (by a known value) if the maximal gap Δ_s between adjacent samples is lower than $1/2M$, which represents the Nyquist limit [75]. In a WSN setting, the sampling locations s_j represent the positions of nodes actively sampling and reporting data. Thus, if the sensor nodes are assumed to have a "virtual" sensing range $R_s = \Delta_s/2$, guaranteeing the fulfillment of both the above mentioned conditions requires providing 1-coverage of the segment $[0, 1]$. Indeed, as shown in figure 4.3 and discussed in [205], if 1-coverage is guaranteed with sensing range $R_s = \Delta_s/2$, then the maximal gap between a node and its closest neighbors is $2R_s = \Delta_s$.

In the 2-dimensional case, the geometry of the problem is more complex. However, as detailed in appendix A, it is still possible to bound the condition number of the matrix T by providing 1-coverage of the region of interest. Assuming the sensor field f to have equal bandwidth in both the x and y directions, the reconstructing polynomial p to look for has order M in both directions. The formulation of the least square problem 4.1 is still valid, provided the sampling locations $s_j = (x_j, y_j)$ belong to the unit square $[0, 1]$x$[0, 1]$, and the weights w_j

[4] The condition number of a matrix A is defined as the ratio between the norm of the matrix and the norm of its inverse, i.e., $c(A) = \frac{\|A\|}{\|A^{-1}\|}$. For the computation of the condition number, usually the L_2 norm is used.

[5] For a given value of M, the condition number of the matrix T is smallest ($= 1$) when the sampling geometry is a uniform grid. Thus, to improve the quality of the reconstruction, the sampling points should be as uniformly spaced as possible.

are set as shown in appendix A. In this setting, the coefficients of the reconstruction polynomial can still be computed by solving system 4.2, where T and b are a $(2M+1)^2 \times (2M+1)^2$ square matrix and a vector with $(2M+1)^2$ entries, respectively. For the matrix T to be invertible, the number of collected samples r must be at least $(2M+1)^2$. This condition is necessary, but not sufficient, to guarantee the invertibility of T. If the sampling locations $s_j, j = 1, \ldots, r$ provide 1-coverage of the square $[0,1] \times [0,1]$ with $R_s = \Delta_s/2$ and $\Delta_s < ln2/(4\pi M)$, then T is invertible and its condition number is bounded (by a known value).

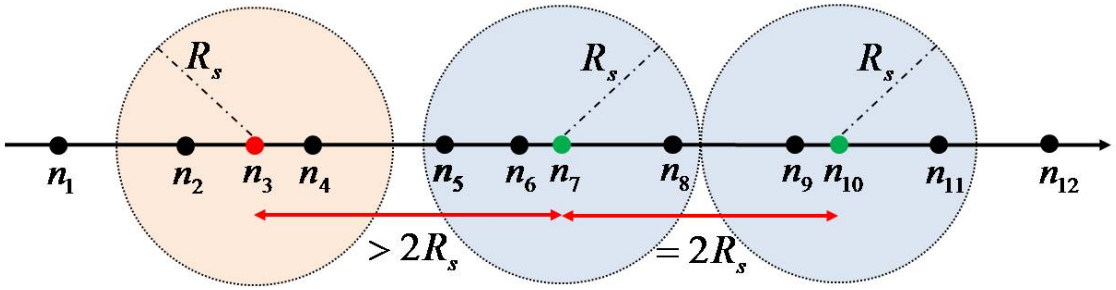


Figure 4.2.: Example of sensor nodes deployment in the 1-dimensional case. If the nodes share the same value of the sensing range R_s, and the deployment area is 1-covered, then the maximal distance between a node and its farthest active neighbors is $2R_s$.

Summarizing the considerations reported above, sampling geometries that provide 1-coverage of the RoI also allow for an efficient and robust execution of the ACT. The value of the "virtual" sensing range R_s of the nodes n_j located at positions $s_j, j = 1, \ldots, r$, depends on the bandwidth of the sensor field of interest and the extension of the RoI. If the RoI is the unitary segment, then $R_s = \Delta_s/2$, with $\Delta_s < 1/2M$ and the number of collected nodes r is at least $2M+1$. If the nodes are deployed over the unit square, then $R_s = \Delta_s/2$, $\Delta_s < \ln 2/(4\pi M)$, and $r \geq (2M+1)^2$ [75]. An appropriate scaling allows to apply these results also to segments or squares of arbitrary dimensions [75, 181].

In the context of our work, we assume an estimation of the value of the spatial bandwidth of the signal, and, thus, of Δ_s, to be known. Alternatively, Δ_s represents the maximal achievable spatial resolution. In this case, the multilevel version of the ACT [75, 169] can come into play to provide a "best-effort" signal reconstruction starting from the available samples. In either case, the problem of providing for appropri-

ate sampling geometries can clearly be restated as a coverage problem. Thus, to design our sensor selection strategy for field reconstruction applications we draw upon existing work on coverage preserving algorithms. In particular, we resort to the CCP protocol by Xing et al. [205], already cited in section 4.2.2, and investigate its adaptability to our application scenario. This choice is motivated by the fact that CCP outperforms other state-of-the-art coverage protocols both in terms of communication overhead and of number of nodes that need to be active to provide for the desired level of coverage [205]. Therefore, CCP appears as one of the most efficient available protocols to generate coverage preserving sampling geometries in WSNs, and, thus, constitutes an ideal starting point for our investigations. In the following section 4.4, we provide a more detailed description of the main features of CCP and discuss some of its limitations.

4.4. The Coverage Configuration Protocol and its Use in Field Reconstruction Applications

Nodes running CCP [205] are assumed to wake up periodically and advertise their presence using HELLO messages. After wake up, a node enters (and stays for a time T_l) in the LISTEN phase, during which it collects information about the presence, position and state of its neighbors by listening to their HELLO messages. Some of the neighbors are likely to have been selected for sensing before and therefore to be in ACTIVE state. These nodes must advertise their activity by sending HELLO messages with high enough frequency. After completing the LISTEN phase, the node computes if its sensing area (i.e., the discus centered at the node and having radius R_s) is already covered by active neighbors. If yes, the node can go back to sleep. Otherwise, it enters a JOIN phase in which it persists at most until a timer T_j expires. The node decides to enter the ACTIVE state if none of its neighbors has advertised itself as active before T_j expires or if its sensing area is not covered by its active neighbors. If, on the contrary, these neighbors are found to be able to cover the sensing area of the node, it can go back to sleep. While in ACTIVE state, nodes continuously collect messages from their neighbors and accordingly reconsider their state. Possibly, they can decide to enter the WITHDRAW state if they realize that coverage is guaranteed also without their contribution. Before abandoning the ACTIVE state, however, nodes wait for a timer T_w to

expire.

A crucial factor influencing the practical implementation and performance of CCP is the choice of adequate values for the timers T_{join} and $T_{withdraw}$. In [205], the authors suggest to randomize the values of these timers to prevent collisions among nodes concurrently deciding about joining or withdrawing the set of active nodes. Using this *random* strategy, the values of the join and withdraw timers $T_{join}(i)$ and $T_{withdraw}(i)$ of a node n_i are drawn at random from a uniform distribution between 0 and T_{join}^{max} or $T_{withdraw}^{max}$, respectively. T_{join}^{max} and $T_{withdraw}^{max}$ represent the maximal values allowed for the join and withdraw timers. In [205] it is suggested to set the values of T_{join}^{max} and $T_{withdraw}^{max}$ according to the network density. In particular, in denser networks nodes should be given more time to collect ACTIVE or WITHDRAW messages from their (crowded) neighborhoods and, thus, the values of the T_{join}^{max} and $T_{withdraw}^{max}$ timers should be accordingly increased. The authors of [205] further suggest that the expiration time of the join and withdraw timers should be ideally linked to the "utility" of a node for the sensing task. In particular, nodes covering more uncovered area should be assigned shorter join timers. However, the definition of proper heuristics to rank the relevance of a sensor has not been further investigated. In the next section 4.5 we propose different strategies to provide for this ranking and, thus, allow for a more effective determination of the values of the timers $T_{join}(i)$ and $T_{withdraw}(i)$ of a node n_i.

Properly setting the values of the timers $T_{join}(i)$ and $T_{withdraw}(i)$ becomes an even more critical issue when the CCP is used in field reconstruction scenarios, instead of for the surveillance and target tracking applications it was originally designed for. Indeed, CCP aims at providing continuous and complete coverage of the RoI over time. To this end, inactive nodes wake up frequently to listen to messages from their neighbors, which may become active or inactive at any instant. Since nodes communicate their joining of or withdrawing from the set of active nodes through a broadcast message, the communication overhead of CCP grows significantly with the number of state changes. In particular, in its very initial phase CCP may activate a number of nodes that is significantly higher than strictly necessary for providing coverage. The protocol can then fix this problem by making nodes turn inactive at a later stage and thus amortize over time this initial overhead. But if CCP is used to provide a sampling geometry for field reconstruction, the relative cost of activating a high number of nodes in its initial stage

becomes far more significant. Indeed, in typical field reconstruction scenarios the network does not operate continuously, but in a round-based fashion. All sensor nodes wake up at predefined intervals Δ_t, perform sampling and, possibly, data communication and go back to sleep. Therefore, although the usage of CCP may still be useful, its cost will be dominated by the number of nodes that decide to become active and then withdraw in the very first phase of the execution of the protocol. Additionally, each time a node receives a JOIN or WITHDRAW message from a neighbor, it must perform computation to establish whether its sensing area is covered by its currently active neighbors or not. If nodes change their state frequently (for instance due to instable links and thus missing notifications) the computational overhead of the protocol rises quickly. Furthermore, if the WITHDRAW notification of a node n_i is lost, part of the network may remain uncovered, since neighbors of the node n_i may still count on its coverage and erroneously decide to withdraw. Thus, limiting the number of potential withdraws implicitly enhances the reliability of the protocol.

These considerations show that, even if CCP can be used as a sensor selection strategy in field reconstruction applications, some of its features may become drawbacks to take into account. In particular, in order to reduce the protocol overhead, it is crucial to minimize the number of nodes becoming active during the inital phase of its execution. To this end, it is particularly important to properly set the values of the timers $T_{join}(i)$. In the next section 4.5, we introduce new heuristics to properly set the values of these timers. In section 4.6, we show that our approach allows to reduce the control overhead of CCP with respect to the random strategy suggested in [205] by more than 10%, on average.

4.5. Adaptive Sensor Ranking

The considerations reported in the previous section show that, for the purpose of field reconstruction, the sampling pattern offered by a WSN should provide for 1-coverage of the RoI, with an appropriate, data-dependent value of the sensing range R_s. An appropriate sensor selection strategy should therefore aim at providing coverage while minimizing the total number of active nodes. Thus, CCP, or other coverage preserving protocols, can be used in this context to provide for good sampling geometries. However, as we also pointed out in section 4.4,

the communication and computational overhead of such protocols is all but negligible, especially if used to generate a short-lived sampling configuration. Using CCP as our reference coverage preserving protocol, we now suggest a set of heuristics that allow to improve its performance. In particular, we propose different strategies to rank the utility of a node n_i for the sensing task and, thus, to determine the value of its activation timer $T_{join}(i)$. Our ranking method aims at extending the CCP protocol presented in [205] and, as we show in section 4.6, allow to improve its performance.

4.5.1. Sensor Ranking Based on Local Densities

As mentioned in [205], the amount of uncovered area a node n_i is able to cover can represent a measure of its relevance to the sensing task. An estimation of this amount could be well approximated by the area of the Voronoi cell of the node [9]. For instance, several field reconstruction algorithms weight the influence of single samples on the global reconstruction using the area of the correspondent Voronoi cells [75, 126]. However, distributed computation of the Voronoi cells in a sensor network requires knowledge of the neighborhood of a node over several hops and an overall high messaging and computational overhead [15]. On the other hand, a centralized computation has well-known drawbacks in terms of scalability and capacity to adapt to changing network topology.

As also shown in [75], a crude estimate of the value of the area of a Voronoi cell is often sufficient to weight the importance of single samples for the field reconstruction process. In particular, the local density of the samples can provide for an estimation of the desired individual weights. Similarly, it is possible to rank the relevance of a node for a sensing task depending on the number of nodes within its communication or sensing neighborhood. Indeed, a node in a low density neighborhood has a higher probability to become active with respect to nodes with a very high number of neighbors. Accordingly, the values of the join timer $T_{join}(i)$ of node n_i could be set as proportional to the local density, so that nodes with fewer neighbors will be the first to decide upon their activation. This method is particularly simple to implement since it only requires the knowledge of the number of neighbors of a node n_i. On the other hand, it does not allow for a fine-grained ranking since nearby nodes will likely have the same number

of neighbors and thus activate simultaneously. Thus, in order to avoid a high number of collisions, this method should be properly refined, possibly including some form of randomization. Furthermore, if the communication range R_{tx} is significantly larger than the desired sensing range R_s, the resulting sensor ranking may be misleading. In this case, it is still possible to consider only those neighbors that lie within the sensing range R_s of the node n_i. To this end, however, the distance of all neighbors to n_i must be known. We will refer to these two strategies for determining the sensor ranking as *density (C)* and *density (S)*, where the C and S letters indicate that only neighbors within the communication or the sensing range are considered, respectively.

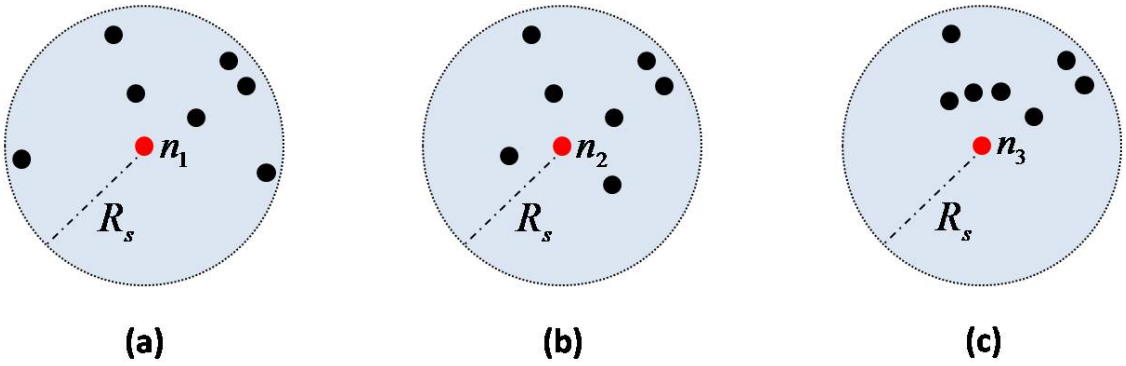


Figure 4.3.: Possible configurations of the sensing neighborhood of a node.

The major drawback of a ranking based on local density consists in the fact that it does not take into account the actual geometry of the neighborhood of the node under consideration. Figure 4.3(a) and 4.3(b) show an example of two nodes n_1 and n_2 having both seven neighbors within their sensing range. If ranked by local density, n_1 and n_2 would be assigned the same rank. However, their actual utilities for the sensing task differ significantly. Indeed, the sensing area of node n_1 would remain uncovered even if all its neighbors were active. Thus, it must become active irrespectively of the decisions of its neighbors and the ideal value of its activation timer is $T_{join}(1) = 0$. On the contrary, three out of the four nearest neighbors of node n_2 can completely cover its sensing area, if they become active before $T_{join}(2)$ expires. Therefore, the value of the activation timer should be set so as to give n_2 enough time to observe whether its nearest neighbors becomes active or not. Clearly, the final outcome depends on the actual geometry of the whole network, but it is reasonable to assume that nodes with further away

neighbors are more likely to become active anyway and, thus, should be given shorter activation timers. In section 4.6, we show that *density (C)* performance gives poor performance, at least if $2R_s \leq R_{tx}$. Considering only neighbors within the sensing range allows for some improvements, but only for low values of the average network density.

4.5.2. Sensor Ranking Using Inverse Distance Weighting

To amend for the above mentioned drawbacks of a ranking based on the local density, we suggest here an alternative ranking method. In particular, we take into account, for the computation of the rank of a node n_i, also the relative distances between the node n_i and its (sensing) neighbors. Intuitively, the absence of nearby neighbors should make the rank of a node increase, signalling that the node has a high probability to be required to become active. On the contrary, the presence of neighbors at very close distance from the node should make its rank decrease, and thus the value of its activation timer increase. To this end, each neighbor n_j of a node n_i can be assigned a weight ϕ_{ij} that is inversely proportional to its Euclidean distance d_{ij} from the node n_i. Using the values ϕ_{ij} it is then possible to compute a weighted local density Ψ_i of the node n_i that represents its ranking for the determination of the activation order.

Inverse distance weighting (IDW) is a technique often used in field reconstruction algorithms to determine the appropriate weights of individual samples [126, 172]. In this context, the weights can be computed using an arbitrary function ϕ and an appropriate normalization, as shown in [172]. Using IDW for ranking the priority with which sensor nodes should activate, however, poses an additional problem. Let us consider again figure 4.3. Node n_3 has, exactly as node n_2, four nearest neighbors all at the same distance. In the first case, however, the four neighbors span a rather small circular sector having the node at its center. In the latter case, on the contrary, the four neighbors are distributed nearly "uniformly" around the node n_2. A method based on IDW would assign this two nodes the same rank, although node n_3 has a higher probability to be required to become active than node n_2. To cope with this problem, we take into account not only the number and distance of the neighbors but also information about their relative position to each other.

Combining the above reported considerations, we define the ϕ_{ij} and

Ψ_i as follows. First, the neighbors are divided into N_{sets} sets $S_{ik}, k = 1, .., N_{sets}$. If the network is deployed on a line (1-dimensional case), the node n_i can assign each neighbor n_j to its "left" ($|s_i| >= s_j$) or "right" (($|s_i| < s_j$) neighborhood, thus $N_{sets} = 2$. If the nodes are deployed on a plane (2-dimensional case), the sets correspond to $N_{sets} = 4$ circular sectors spanning the circle centered on the node and having radius R_s. If all sets are non-empty, the node computes, for each neighbor n_j, an inverse distance weighting function $\phi(d_{ij}) = \phi_{ij}$. There exist several different functions that can provide for a proper IDW metric [172], but we consider here a simple linear weighting and set

$$\phi_{ij} = 1 - \frac{d_{ij}}{\alpha R_s}. \tag{4.3}$$

In both the 1- and 2-dimensional case, we found that setting $\alpha = 1$ provides for highest performance gains. For the following considerations we therefore assume $\alpha = 1$. This also implies that only those neighbors whose distance d_{ij} to n_i is strictly smaller than R_s, are considered for computing the rank of the sensors, since the weights ϕ_{ij} must be positive values.

The contribution of all neighbors in a sector k is then computed as:

$$\Psi_{ik} = \frac{1}{1 + \sum_{j=1}^{N_{ik}} \phi_{ij}}, \tag{4.4}$$

where N_{ik} is the number of neighbors included in the set S_{ik}. Ψ_{ik} basically represents the weighted local density of the node n_i, relative to the sector k. We can also rewrite equation 4.4 as:

$$\Psi_{ik} = \frac{1}{\sum_{j=0}^{N_{ik}} \phi_{ij}} \tag{4.5}$$

where the "neighbor" of index $j = 0$ corresponds to the node n_i itself and, thus, $d_{i0} = 0$. For each set k the value Ψ_{ik} represents a measure of the importance of the node n_i to "cover" the region spanned by the set k. For an empty set this value is clearly equal to 1. An appropriate aggregate (e.g., minimum or average) of the Ψ_{ik} then gives the desired ranking Ψ_i of the node n_i. Our simulation results showed the average to be the more appropriate aggregation function. Thus, we set:

$$\Psi_i = \frac{1}{N_{sets}} \sum_{k=1}^{N_{sets}} \Psi_{ik} \tag{4.6}$$

The value of Ψ_i asymptotically goes to zero as the number of nodes in each set k increases. If the node n_i has no neighbors in its sensing range, Ψ_i reaches its maximum value 1. The activation timer of the node n_i can then be accordingly set as

$$T_{join}(i) = T_{join}^{max}(1 - \Psi_i). \tag{4.7}$$

We refer to this strategy for setting the activation timers as the *idw* method. In section 4.6 we show that it allows for performance improvements with respect to both the *density (C)* and *density (S)* methods, although it requires only minimal additional computational and memory overhead.

In static networks, however, the *idw* method (as well as the *density (C)* and *density (S)* strategies) has an important drawback. If nodes do not change their position or do so only infrequently with respect to the temporal sampling rate of the network, the ranking of the nodes given by the above described methods will be the same in every round. This means that, up to little differences due to possible communication failures, the same set of nodes will become active in every round. Clearly, this hampers the possibility to balance the overall energy consumption due to participation in sensing. To cope with this problem, we resort to randomization, as explained in the next section.

4.5.3. Inverse Distance Weighting and Random Sampling

The Ψ_i defined in equation 4.6 can be interpreted as the probability of the node n_i to become active in a given sampling round. Indeed, if n_i has no neighbors closer than αR_s, it is forced to become active and, thus, its probability of activation is 1. In the other extreme case in which it has N_i neighbors at its same position (thus, $d_{ij} = 0, j = 1, \ldots, N_i$), its probability of activation will simply be $1/N_i$. In other words, the node should ideally become active once every N_i rounds since its shares the "sensing responsibility" for its sensing area with N_i "identical" peers. In all intermediate cases the presence of a neighbor n_j at a certain distance d_{ij} will "relieve" the node n_i of an amount of its sensing responsibility that is inversely proportional to the distance d_{ij}.

To mitigate the load balancing inefficiency of the *idw* strategy, we thus suggest to use the value of Ψ_i as a probabilistic ranking of the node. In other words, the value of the timer $T_{join}(i)$ of a node n_i can

be set as:

$$T_{join}(i) = T_{join}^{max} \cdot p(\Psi_i). \tag{4.8}$$

where $p(\Psi_i)$ is a value drawn at random from a uniform distribution between 0 and $1 - \Psi_i$. In the following section 4.6, we show that this strategy, dubbed *idw random*, not only allows to better balance energy consumption due to sensing across the nodes in a network, but can also outperform other methods in terms of number of active nodes. However, also using the *idw random* strategy nodes with higher ranks are required to become active more often than low-rank nodes. Therefore, high rank nodes will likely drain their batteries sooner than others and, thus, the load balancing problem discussed above, although mitigated, still persists. As discussed in [151], an effective way to preserve the energy of high rank nodes consists in limiting their participation in other network activities, like, e.g., data routing. In section 4.8, we show how the sensor ranking strategies presented above can be leveraged for balancing nodes participation in sensing and routing.

4.6. Experimental Results

We report experimental results showing the ability of the heuristics discussed in the previous section to provide for good sampling geometries while limiting the number of active nodes. Our experimental setup and the metrics used to measure performance are described in detail in the next section 4.6.1. We then report our simulation results for the 1- and 2-dimensional cases in sections 4.6.2 and 4.6.3, respectively.

4.6.1. Experimental Setup

We consider a network of N_{tot} nodes deployed uniformly at random over either a segment of length L_x (1-dimensional case) or a rectangular region of sides L_x and L_y (2-dimensional case). We assume that all nodes have a fixed communication range of R_{tx} meters and that we can rely on the communication model discussed in section 2.3. Further, we assume that the application requires to retrieve data with a spatial resolution of Δ_s meters and we thus set $R_s = \Delta_s/2$.

We implemented the CCP protocol in Matlab, a well-known simulation environment already described in section 2.2.3. We compare the performance of the CCP coverage protocol using different methods to

determine the values of the timers T_{join}. In particular, we consider the heuristics *random*, *density (C)*, *density (S)*, *idw*, and *idw-random*. As discussed previously, the *random* strategy, proposed in [205], lets the value of the timers T_j be independently drawn from a uniform distribution between 0 and T_{join}^{max}. The second and third method set the values of the timers depending on the local density, computed considering all neighbors within the communication or sensing range, respectively. The higher the number of neighbors, the higher the corresponding value of the timer T_{join}, in either case. The *idw* strategy ranks the sensor nodes according to the metric defined in equation 4.6 and makes nodes with higher rank to activate first. Finally, the *idw-random* combines both the ranking provided by the *idw* strategy and an appropriate randomization, as discussed in section 4.5.3. We assume the value of T_{join}^{max} to be fixed a priori and available to all nodes.

In all our experiments, we consider the following setting. At predefined time intervals $k\Delta_t, k = 1, 2, \ldots$, all nodes in the network wake up and collect information about their neighborhood for a time interval T_l, which sets the length of the initial *listen phase*. After T_l expires, we can thus assume every node n_i to hold an updated list of its neighbors, along with their positions. Then, the network enters the *activation phase* in which each node determines whether it must become active or not. The order in which nodes decide upon their activation depends on the values of the timers T_{join} and is therefore determined by the strategy used to set these values. After the timer T_{join}^{max} expires, all nodes are assumed to have decided upon their activation and the network can thus enter the *withdrawal phase*. In this phase, nodes possibly withdraw from the set of active nodes if they found their active neighbors to be able to cover their sensing area. The order in which nodes decide upon their withdrawal is the same order used in the activation phase. This can be easily implemented by making nodes start a withdraw timer $T_w = T_{join}^{max}$ immediately after they make the decision to become active. The total duration of the listen, activation and withdraw phases is, therefore, $T_{tot} = T_l + 2T_{join}^{max}$. During this interval sensor nodes must keep their radio circuitry powered on to receive and send notifications about possible activations and withdrawals. Therefore, the value of T_{tot} should be kept small, so as to limit energy consumption. Also, to save additional energy, nodes that do not become active during the activation phase can immediately switch off their radio and go back to sleep, unless they are required to remain active for routing purposes. On this

regard we should also recall that, if the communication range is at least twice as big as the sensing range, connectivity can be guaranteed by the set of active nodes [205]. In this case, inactive nodes can switch off their radios as soon as their activation timer expires without further restrictions.

We assume the network can rely on one of the routing protocols available in the literature to report data back to the central server [1,3]. However, we require information about neighbors' positions to be available in the routing table. With these assumptions, the listen phase described above simply reduces to an access to the information available at the routing layer. Assuming a stable network, this information does not change frequently, and thus the rank of the nodes will remain stable across successive sampling rounds. A long listen phase that can guarantee sufficient information about the neighborhood to be collected, therefore, is only necessary at the very beginning of network operation. Later, we can assume $T_l = 0$ and make the node use the previously computed value of the sensor rank as the current value. Before going back to sleep again, each node can then update the value of its rank by including the contribution of newly added neighbors or discarding that of missing ones. This mechanism clearly introduces some latency in the ability of the node to evaluate its sensing rank, but allows to simplify and increase the energy efficiency of our modified version of CCP. The initial value of T_l can be set proportionally to the average density of the network, as suggested in [205].

Making the nodes wake up at predefined time intervals and operate using a shared timer T_{join}^{max} requires the network to be, at least loosely, synchronized. To this end, we assume one of the protocols known in literature to be applied [159]. As for localization, we assume the node can retrieve their position autonomously, for instance using a GPS sensor, or through one of the available localization algorithms [29, 113, 149].

The main goal of introducing new heuristics to determine the activation timers of the nodes is that of reducing the total number of nodes that become active during the activation phase. This allows to reduce the communication overhead of the CCP protocol, and thus its energy consumption, since each node n_i that becomes active broadcasts a corresponding notification to its N_{Ci} neighbors. Thus, if C_{tx} and C_{rx} represents the cost of transmitting and receiving a packet, respectively, the communication cost due to the activation of a node n_i

is $C_{Ai} = C_{tx} + N_{Ci} \cdot C_{rx}$. If S_{ANbw} is the set of the N_{ANbw} nodes that activate before the withdraw phase starts, then the total cost of the activation phase in terms of communication is given by:

$$C_A = \sum_{j \in S_{ANbw}} (C_{tx} + N_{Cj} \cdot C_{rx}) = N_{ANbw} C_{tx} + C_{rx} \sum_{j \in S_{ANbw}} N_{Ci}. \quad (4.9)$$

Similarly, if S_{WN} is the set of the N_{WN} nodes that decide to leave the set of active nodes during the withdraw phase, the total cost of this phase in terms of communication is given by:

$$C_W = \sum_{j \in S_{WN}} (C_{tx} + N_{Cj} \cdot C_{rx}) = N_{WN} C_{tx} + C_{rx} \sum_{j \in S_{WN}} N_{Cj}. \quad (4.10)$$

Thus, the total cost in terms of communication due to the use of the modified version of the CCP protocol presented above is

$$C_{CCP} = C_A + C_W. \quad (4.11)$$

In equation 4.11 we neglect the cost of the listen phase. Indeed, neighborhood discovery must be performed anyway for routing purposes. Thus, we focus on the additional overhead due to the action of the CCP protocol only, which is expressed by equation 4.11. The costs C_{tx} and C_{rx} can be set using, for instance, the method proposed in [84]. However, since we are concerned only with the number of packets that are transmitted and received, we can simply assign a unitary value to both C_{tx} and C_{rx}. On some platforms the cost related to the transmission of packet may be higher then the cost of receiving one[6]. However, for several other nodes, including the Tmote Sky, our reference platform, $C_{tx} = C_{rx}$ holds and we thus report our results under this assumption.

During the activation and withdraw phase, sensor nodes must keep their radios in idle listening, since they cannot predict the time instants at which neighbors will possibly send their activation or withdraw beacons. This additional source of energy consumption is not included in equation 4.11 since it is common to all the strategies considered to set the activation timers. However, reducing the number of nodes becoming active in the activation phase allows to switch off more nodes earlier,

[6] For instance, the transceiver of the TinyNode 184 draws 25mA of current in transmit mode and only 3mA in receive mode (nominal values) [173].

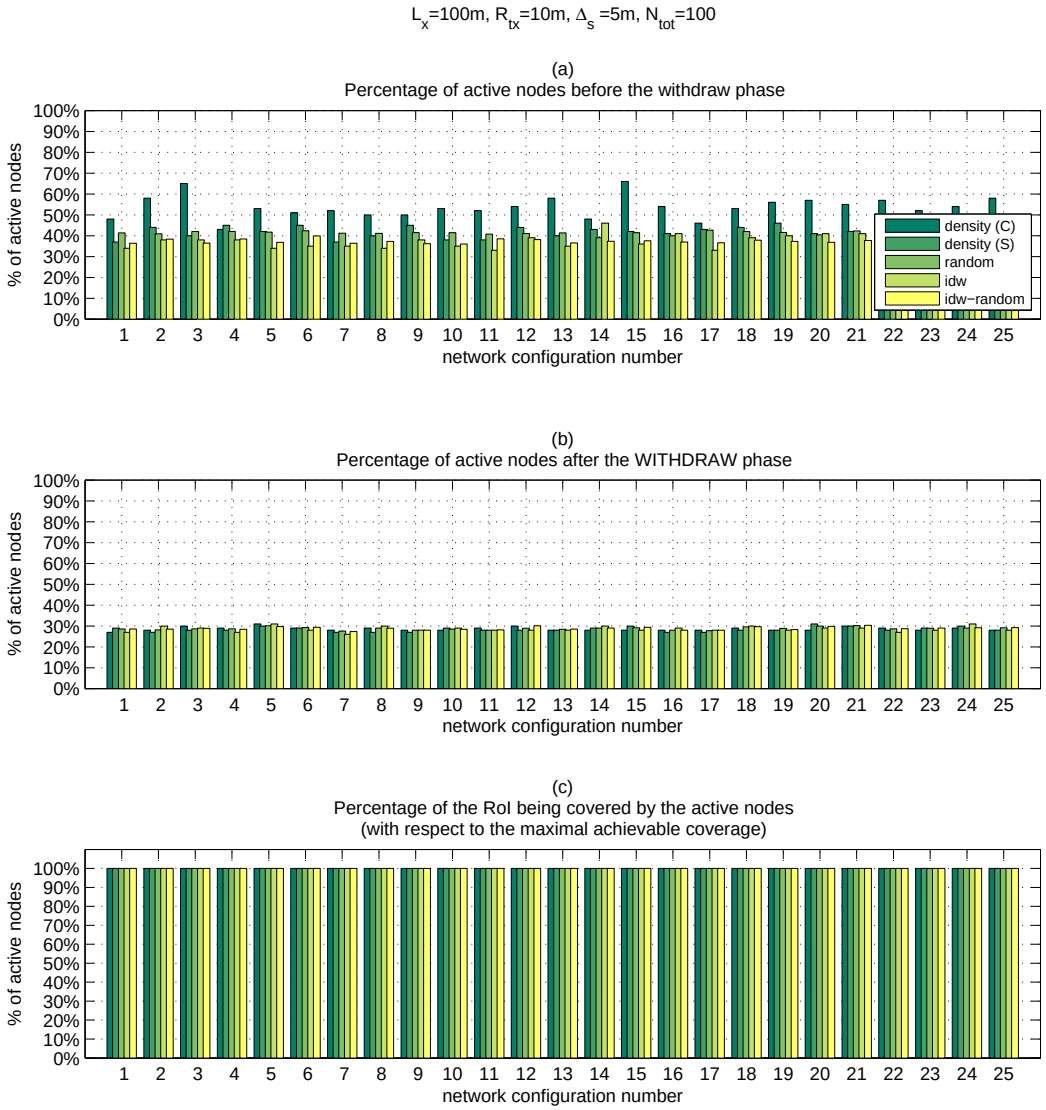


Figure 4.4.: Number of active nodes before (a) and after (b) the withdraw phase, and percentage of the RoI being covered by the active nodes (c). Results in (a) and (b) are in percentage with respect to the total available number of nodes N_{tot}. Results in (c) are in percentage with respect to the coverage achievable by the activation of all the N_{tot} available nodes. Experimental setting (1-dimensional case): $L_x = 100m$, $R_{tx} = 10m$, $Delta_s = 5m$, $N_{tot} = 100$.

and, thus, to reduce the amount of time these nodes must persist in idle listening. Using platforms with low energy consumption in receiving mode, like the TinyNode 184 [173], would allow to significantly reduce the impact of idle listening on the total energy consumption of CCP. To this end, we should simply accordingly set the costs C_{tx} and C_{rx} that appear in equation 4.11. As explained above, however, we report our experimental results for the case $C_{tx} = C_{rx} = 1$.

Another parameter that is important to observe is the total number of nodes that remain active after the withdrawal phase, N_{ANaw}. If nodes do not die or run out of batteries while the CCP is operating sensor selection, we clearly have: $N_{ANaw} = N_{ANbw} - N_{WN}$. As we will detail below, the ranking strategies we consider achieve, on average, the same values of N_{ANaw}. On the contrary, they may show consistent differences in terms of N_{ANbw} and N_{WN}. In other words, all strategies allow to achieve complete coverage of the RoI, but at different costs.

To gather statistically significant values of the above described metrics, we run the CCP over 25 different random network configurations. For ranking strategies involving elements of randomness (namely the *random* and *idw-random* methods), we run 25 trials for each configuration.

4.6.2. 1-Dimensional Case

We begin the evaluation of our ranking strategies showing experimental results obtained in the 1-dimensional case. Unless specified otherwise, we consider a RoI of length $L_x = 100m$ and communication range R_{tx} of the nodes equal to $10m$. The virtual sensing range of the nodes is set as $\Delta_s/2$, where Δ_s is the maximum allowed distance between an (active) node and its closest (active) neighbor.

Figure 4.4 shows the performance of all the considered strategies, for 25 different random configurations and when $\Delta_s = 5$ and $N_{tot} = 100$. Figure 4.4(a) shows the percentage of nodes (with respect to the total number of nodes N_{tot}) that become active during the activation phase. As we can see, the *idw* and *idw-random* strategies outperform the other methods in nearly every case, since they require a smaller number of nodes to become active. In particular, they both perform better than the *random* strategy, which was proposed in [205].

Figure 4.4(b) also shows the percentage of active nodes, but after the withdrawal phase has been run. The different strategies require, up to

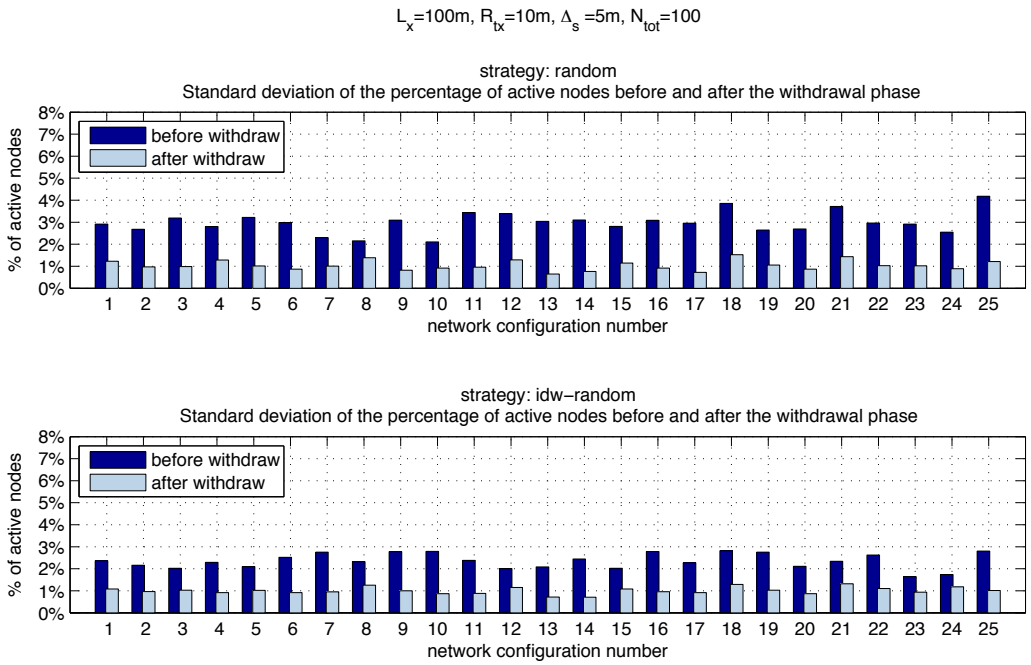


Figure 4.5.: Standard deviation of the number of active nodes before and after the withdraw phase, for the *random* and *idw-random* strategies. Results are in percentage with respect to the total available number of nodes N_{tot}. Experimental setting (1-dimensional case): $L_x = 100m$, $R_{tx} = 10m$, $Delta_s = 5m$, $N_{tot} = 100$.

some small random fluctuations, the same number of nodes to remain active in order to cover the RoI with the required resolution. Thus, the additional overhead caused by the *random*, *density (C)*, and *density (S)* methods with respect to the *idw*, and *idw-random* strategies does not pay off with better performance in terms of number of eventually active nodes.

For the sake of completeness, figure 4.4(c) also shows the level of coverage reached by the set of active nodes. As expected, all strategies reach complete coverage of the RoI, or, better said, the same level of coverage that is achievable by activating all the deployed nodes. Indeed, the values reported in figure 4.4(c) are normalized to the maximum achievable coverage, which for some configurations is slightly smaller than 100% of the RoI.

For the *random* and *idw-random* strategies, the values reported in figure 4.4 have been averaged over 25 runs. To give a feeling of the fluctuations that can affect this average value, we report the standard deviation of the number of active nodes before and after the withdrawal phase for both *random* and *idw-random* in figure 4.5. As we can see, the standard deviation of the *idw-random* strategy is in general smaller

than the corresponding value of the simple *random* method. The overall limited variability of the results across the 25 trials allows to consider representative the average values reported in figure 4.4. Similar considerations apply for the results obtained in experiments run with different parameters, which are discussed below.

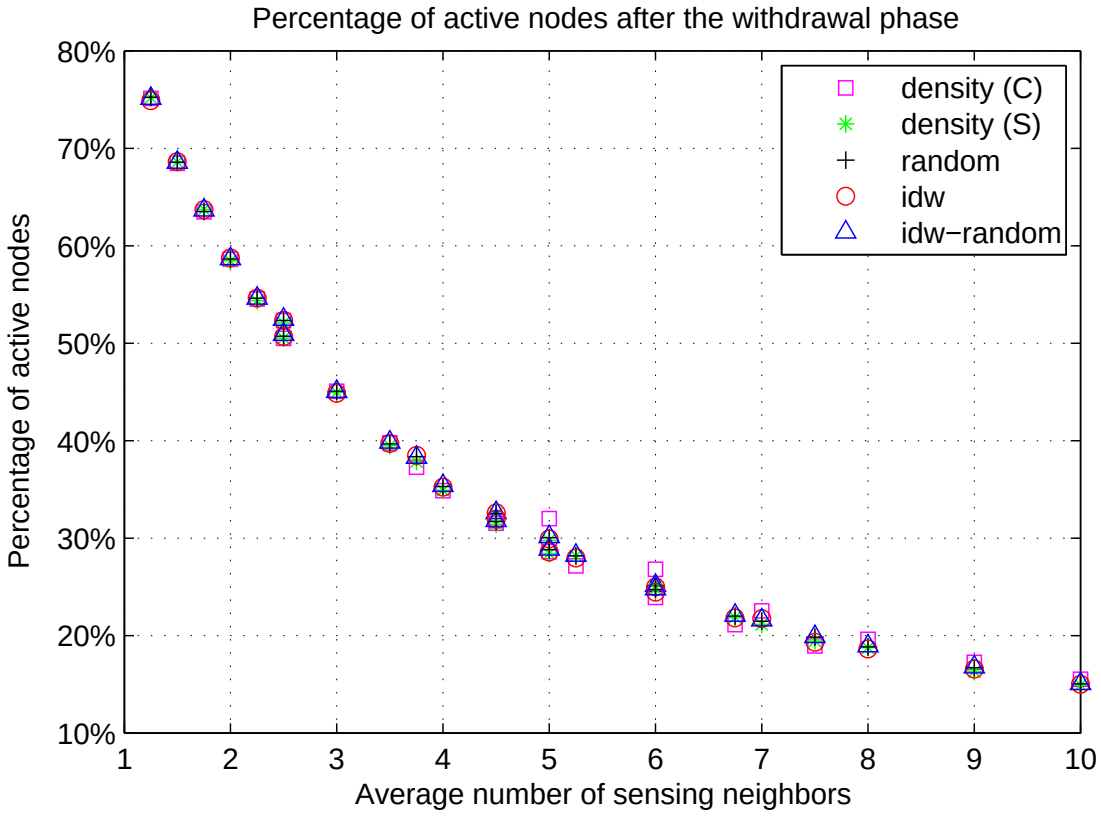


Figure 4.6.: Number of active nodes after the withdrawal phase as the average number of sensing neighbors increases. Results are in percentage with respect to the the total number of available nodes N_{tot}. Experimental setting (1-dimensional case): $L_x = 100m$, $R_{tx} = 10m$, $Delta_s = 2.5, 5, 7.5$, and $10m$, $N_{tot} = 50, 60, 70, 80, 90$, and 100.

Figure 4.4 shows that, for a specific set of parameters, the *idw* and *idw-random* strategies allow to reduce the communication overhead of the CCP protocol. To gain a more comprehensive view of the performance of the 5 different strategies under examination, we must consider a larger spectrum of variability of the crucial parameters, namely Δ_s and N_{tot}. We thus run a series of experiments letting the number of nodes N_{tot} vary from 50 to 100 and, concurrently, the value of Δ_s step from 25% to 100% of the communication range (thus, being $R_{tx} = 10m$, from $2.5m$ up to $10m$). For each experiment, we compute the percentage of active nodes before and after the withdrawal phase, as well as

the total communication overhead as specified in equation 4.11. Since the communication overhead of the different strategies depends upon the local sensing density of the nodes, we display these comprehensive results as the ratio $\Delta_s N_{tot}/L_x$ increases.

Figure 4.6 shows that the number of nodes required to be active after the withdrawal phase, is practically identical for all the considered strategies and irrespective of the local sensing density. This shows that, in terms of number of nodes that eventually remain active, the considered strategies are interchangeable. Therefore, a higher number of active nodes before the withdrawal phase constitutes a net cost in terms of communication overhead, since it does not allow for any performance gain afterwards.

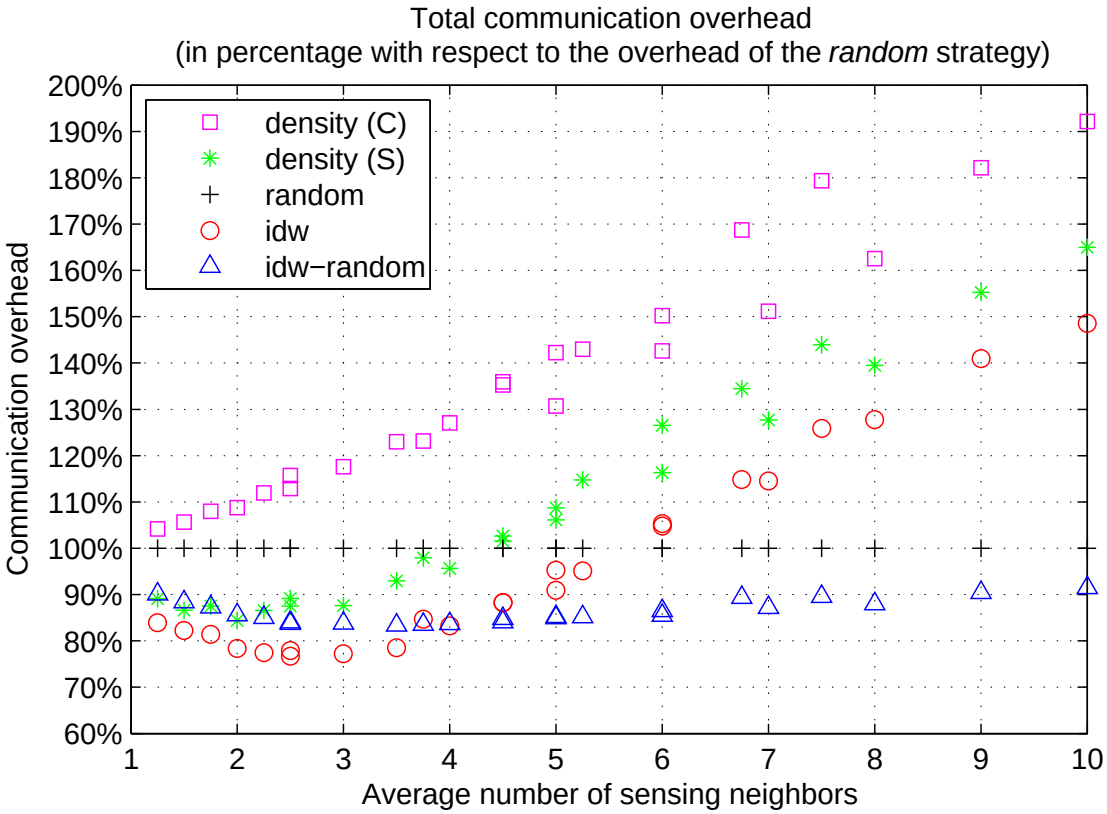


Figure 4.7.: Communication overhead of the CCP using different strategies to set the activation timers of the nodes, as the average number of sensing neighbors increases. Results are in percentage with respect to the overhead of the *random* strategy. Experimental setting (1-dimensional case): $L_x = 100m$, $R_{tx} = 10m$, $Delta_s = 2.5, 5, 7.5$, and $10m$, $N_{tot} = 50, 60, 70, 80, 90$, and 100.

To show the amount of this cost, figure 4.7 displays the total number of messages sent and received by the CCP protocol as the (average)

local sensing density of the network increases. Since we are evaluating the achievable improvements with respect to the *random* strategy proposed in [205], we normalize the results of each experiment to the cost of this strategy. Thus, figure 4.7 shows the achievable gains, or losses, in terms of communication overhead that incur in using a different strategy rather than the *random*.

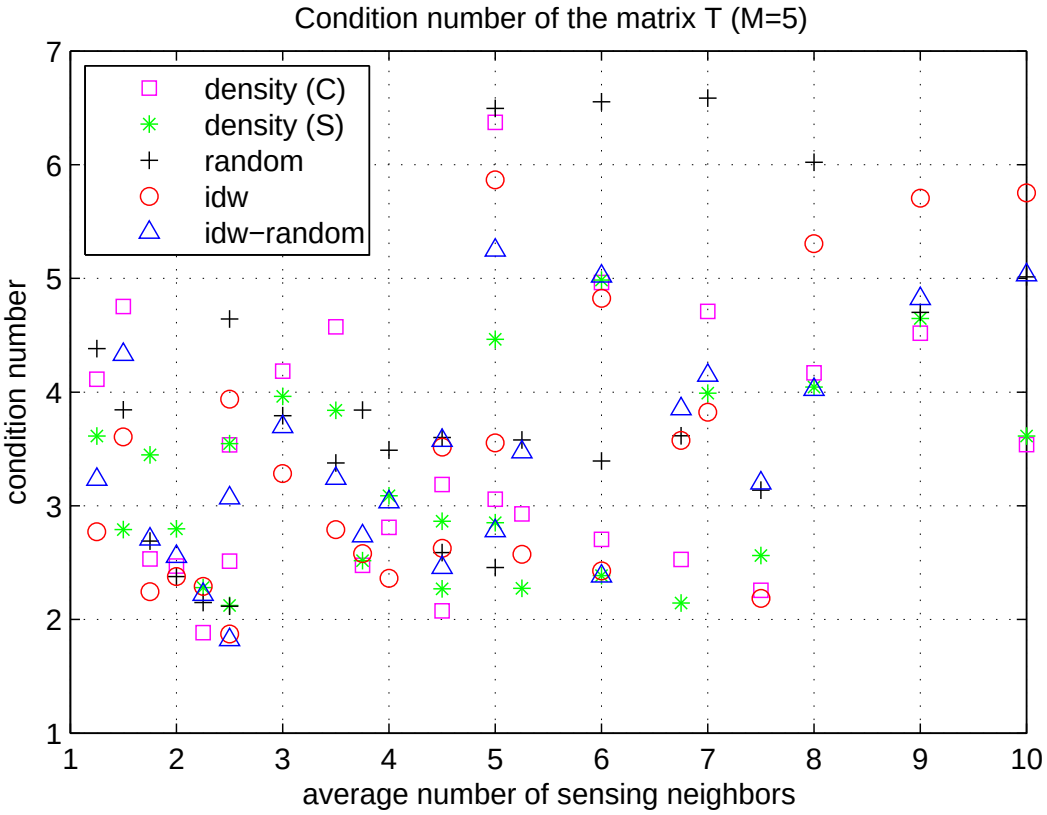


Figure 4.8.: Value of the condition number of the matrix T as the average number of sensing nodes increases (M=5). Experimental setting (1-dimensional case): $L_x = 100m$, $R_{tx} = 10m$, $Delta_s = 2.5, 5, 7.5$, and $10m$, $N_{tot} = 50, 60, 70, 80, 90$, and 100.

As we can see, the *density (C)* strategy performs worse than all the others for every combination of parameters. This is due to the fact that considering neighbors beyond the sensing range of the nodes introduces a sort of "noise" in the ranking. Indeed, the further away the neighbors are, the less they can and should influence the decision of a node to become active. This consideration is also supported by the fact that the *density (S)* strategy performs, at least at low densities, better. As the local density increases, however, the performance of both the *density (C)* and *density (S)* worsen significantly. We should also notice that the *density (C)* and *density (S)* strategy perform differently also as Δ_s

reaches the value $10 = R_{tx}$ since the radius of the sensing neighborhood is $R_s = \Delta_s/2$.

In general, we claim that heuristics based on the local density do not offer satisfactory results due to the reasons discussed in details in section 4.5. However, when the average local density is very low, the *density (S)* strategy can correctly capture nodes in nearly isolated neighborhood as those that must activate first. Already as the average number of neighbors reaches the value of 2, however, the performance of the *density (S)* method starts deteriorating. As the local density reaches the value of 4, the *random* strategy can already outperform the *density (S)* method. On the contrary, the *idw* method demonstrates superior performance with respect to the *random* strategy up to a critical value (~ 6) of the average number of sensing neighbors. Before reaching this critical threshold, idw can save up to more than 20% of communication overhead with respect to the *random* strategy and at least 5% or more with respect to the *density (S)* method. This superior ability with respect to the *density (S)* method is due to the fact that the *idw* considers also the actual geometry of the sensing neighborhood instead of no information at all (as for the random strategy) or its cardinality only (as *density (S)* does). However, as the number of sensing neighbors increases, also the performance of *idw* deteriorates. We attribute this loss in performance to the fact that, as the number of nodes entering in the computation of the rank of a node increases, the absolute values of the rank and the separations among them becomes smaller, and, thus the possibility of assigning a node a misleading ranking increases. In other words, the presence of more nodes creates more "noise" that can hamper a proper discrimination of the rank of the nodes. We believe this drawback could be alleviated by performing inverse distance weighting using a quadratically or exponentially decaying function of the distance. In this way, the weighting would better accentuate the differences between nearby and further away nodes and thus possibly allow for a more fine-grained ranking. Alternatively, randomization can provide for the necessary re-balancing of the rankings. Indeed, figure 4.7 shows that the *idw-random* strategy can outperform the *random* method by $\sim$ 10% or more for every level of the local density.

The above presented results show that our heuristics to set the activation timers T_{join} are effective in reducing the communication overhead of the CCP protocol. Further, we would like to prove that the sam-

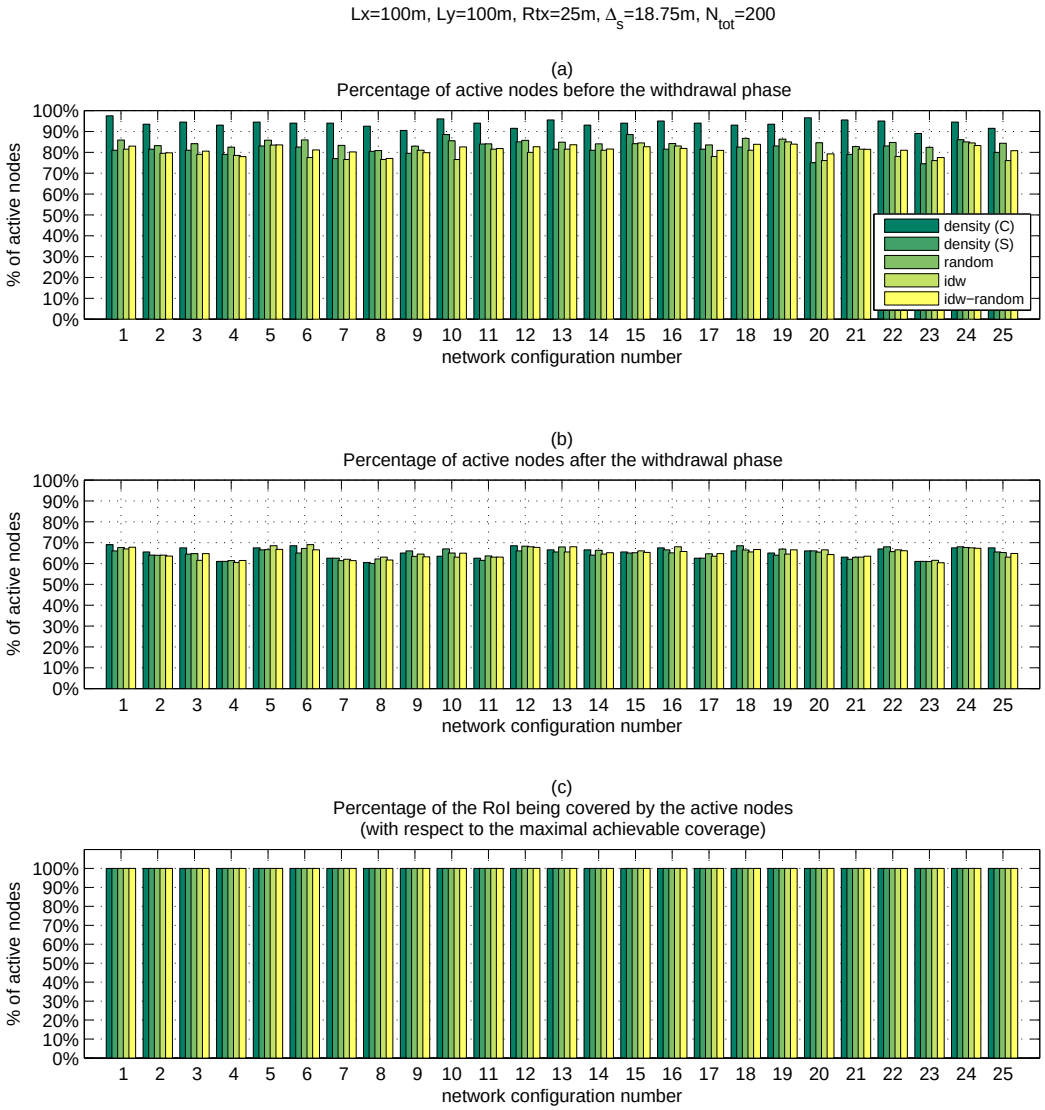


Figure 4.9.: Number of active nodes before (a) and after (b) the withdrawal phase, and percentage of the RoI covered by the active nodes (c). Results in (a) and (b) are in percentage with respect to the total available number of nodes N_{tot}. Results in (c) are in percentage with respect to the coverage achievable by the activation of all the N_{tot} available nodes. Experimental setting (2-dimensional case): $L_x = 100m$, $L_y = 100m$, $R_{tx} = 25m$, $Delta_s = 18.75m$, $N_{tot} = 200$.

pling patterns induced by the set of active nodes on the sensor field to sample, can enable appropriate reconstruction using the ACT method. To this end, we consider again the set of experiments described above and compute, for appropriate values of M, the condition number of the reconstruction matrix T. Figure 4.8 shows the average value of the condition number of T as the average density increases, for $M = 5$. As we can see, all the considered strategies can provide for similar values of the condition number and, thus, for a robust execution of the ACT. This result is not surprising since the value of the condition number $k(T)$ depends on the total number of samples and the maximal distances between adjacent samples. Since, as shown by figure 4.6, all strategies eventually provide for the same number of active nodes, as well as for complete coverage of the RoI, it is perfectly reasonable that they also return sampling geometries that achieve similar values of the condition number of T. However, using the *idw* and *idw-random* strategies allow to reach these results with significantly less overhead with respect to the *density (C)*, *density (S)*, or *random* methods.

Finally, we would like to point out that, although it may look less interesting than the 2-dimensional case, the 1-dimensional case is still of practical relevance. For instance, sensors deployed along a road or railway can be (at least locally) seen as nodes over a 1-dimensional space. Similarly, cars driving on a highway can be represented as mobile nodes deployed on a line, as well as people walking on a street.

4.6.3. 2-Dimensional Case

The results obtained in the 2-dimensional case are qualitative very similar to those previously discussed for the 1-dimensional case. To measure the performance of the different heuristics, we considered again a variable number of nodes N_{tot}, from 200 to 300 deployed over a square region of sides $L_x = L_y = 100m$. The transmission range R_{tx} has been fixed to $25m$ and the value of Δ_s increased from 75% to 100% of R_{tx}.

For instance, figure 4.9 shows the percentage of active nodes before and after the withdrawal phase and the corresponding level of coverage for the case $\Delta_s = 0.75R_{tx}$ and $N_{tot} = 200$. As we can see, the *idw* strategy is, in most cases, the method that allows to minimize the number of active nodes, although for some configurations it is slightly worse than the *density (S)* strategy. For the sake of completeness, figure 4.10 also shows the standard deviation of the two randomized

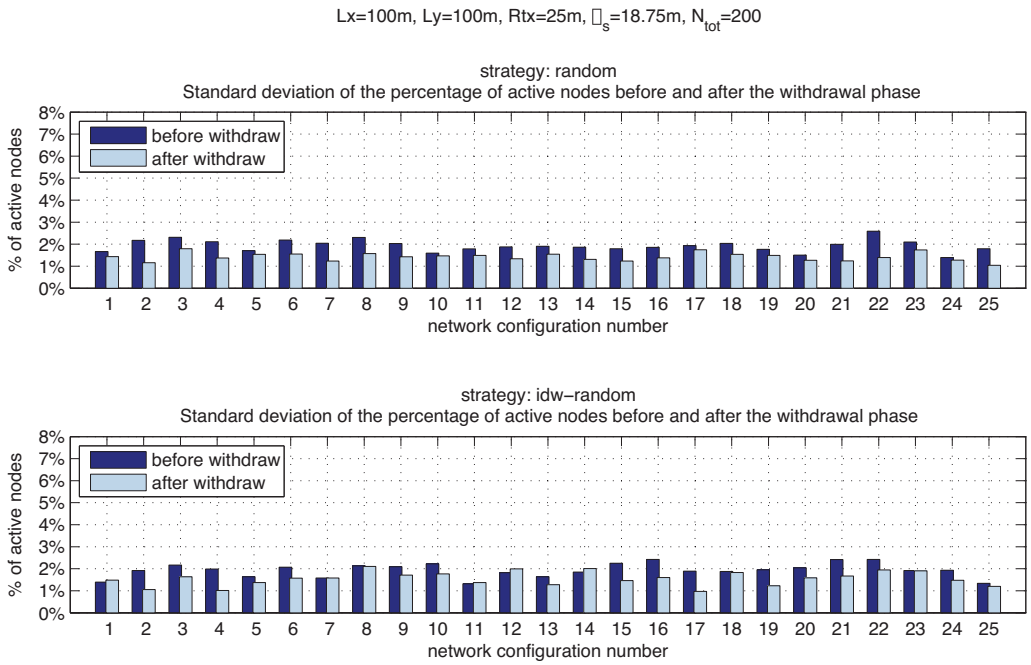


Figure 4.10.: Standard deviation of the number of active nodes before and after the withdrawal phase, for the *random* and *idw-random* strategies. Results are in percentage with respect to the total available number of nodes N_{tot}. Experimental setting (2-dimensional case): $L_x = 100m$, $L_y = 100m$, $R_{tx} = 25m$, $Delta_s = 18.75m$, $N_{tot} = 200$.

strategy corresponding to the results displayed in figure 4.9.

As already seen for the 1-dimensional case, the performance of a specific strategy depends on the local average density of the nodes. Figure 4.11 gives a more general picture of the performance of the different strategies as the local density increases. Similarly as what we observed for the 1-dimensional case, the *idw* and *density (S)* metrics perform well for low densities. As the average number of nodes in a sensing neighborhood increases, however, randomized strategies outperform deterministic methods. Also, we can observe that the *idw-random* strategy requires about 10% less communication overhead with respect to the plain *random* assignment method.

Figure 4.12 also reports, as a function of the average number of sensing neighbors, the number of nodes that eventually remain active after the withdrawal phase. On this regard, as in the 1-dimensional case, the considered strategies have practically identical performance.

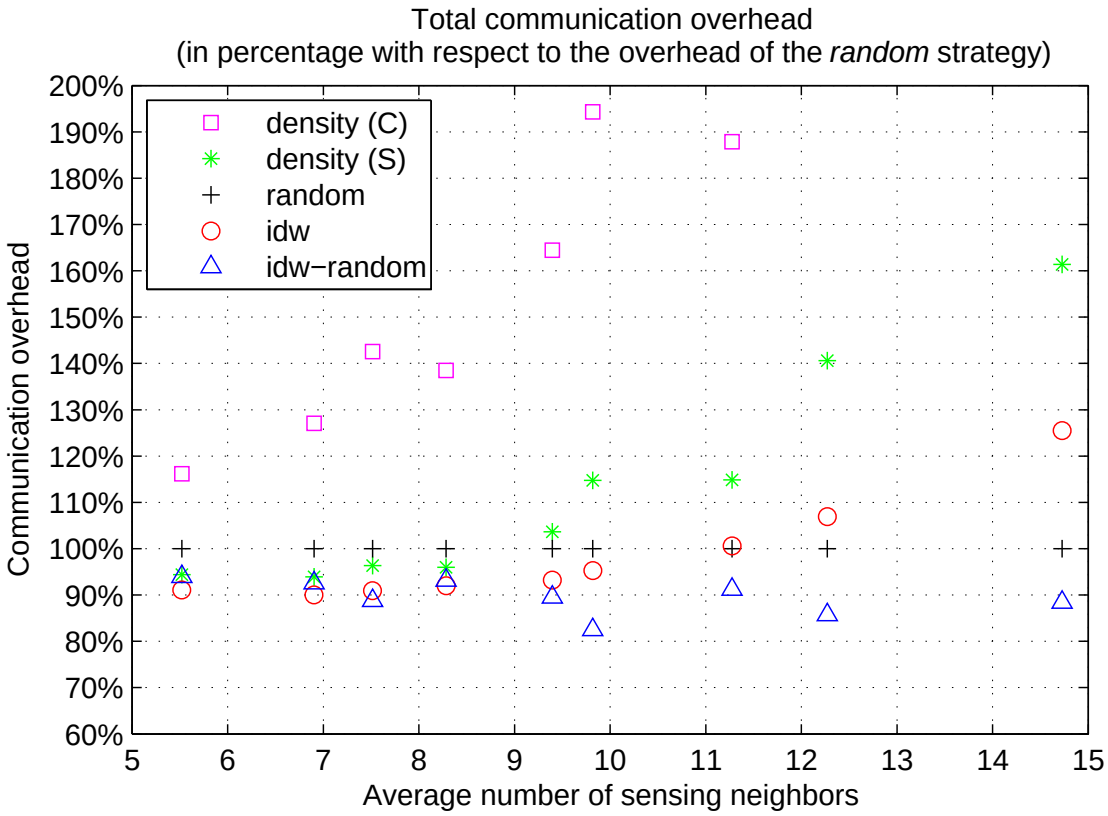


Figure 4.11.: Communication overhead of the CCP using different strategies to set the activation timers of the nodes, as the average number of sensing neighbors increases. Results are in percentage with respect to the overhead of the *random* strategy. Experimental setting (2-dimensional case): $L_x = 100m$, $L_y = 100m$, $R_{tx} = 25m$, $Delta_s = 18.75, 21.875$, and $25m$, $N_{tot} = 200, 250$, and 300.

4.7. Adaptive Random Sensor Selection (ARS)

In the previous section, we have shown that our ranking heuristics are effective in reducing the communication overhead of the CCP protocol. Also, we have observed that techniques based on randomization may often perform comparably, or better, than deterministic, and possibly more complex, strategies. In this section, we further investigate the possibility of using our sensor heuristics to improve the performance of sensor selection strategies based on random node activations. To this end, we first briefly recall the concept of random sensor selection and summarize related theoretical results in sections 4.7.1 and 4.7.2, respectively. We then introduce a new adaptive random strategy (ARS) for sensor selection that leverages our ranking heuristics in section 4.7.3 and provide its evaluation in section 4.7.4.

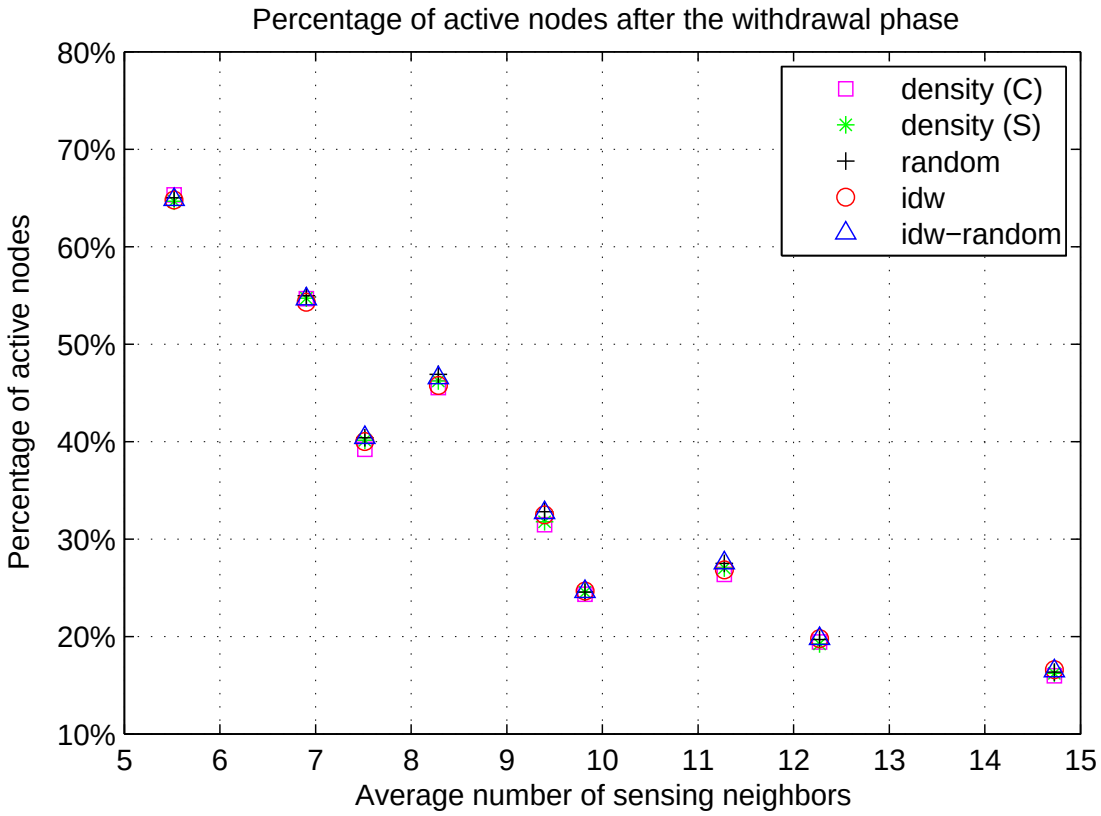


Figure 4.12.: Number of active nodes after the withdrawal phase as the average number of sensing neighbors increases. Results are in percentage with respect to the the total number of available nodes N_{tot}. Experimental setting (2-dimensional case): $L_x = 100m$, $L_y = 100m$, $R_{tx} = 25m$, $Delta_s = 18.75, 21.875$, and $25m$, $N_{tot} = 200, 250$, and 300.

4.7.1. Random Sensor Selection

The probably simplest ways to perform spatial sensor selection in WSNs is to make the nodes randomly decide upon their activation. In particular, each node can be assigned a *probability of activation* p, which can be computed locally or disseminated by a central server. Each time data collection is required, sensor nodes can autonomously decide whether to participate in the sensing task or not by generating a random number ρ between 0 and 1. If $\rho < p$, the node collects and reports data, and it remains idle otherwise. This simple random sensor selection (RSS) protocol clearly requires very little control overhead. Indeed, once the value p has been fixed and disseminated to all the nodes, the protocol must intervene only to update possibly newly added nodes or to opportunely adapt the value of p. Its efficacy is indirectly demonstrated by the results reported in [120], which show that the RSS can perform

comparably, or even better, than other more sophisticated (and costly) sensor selection protocols.

Clearly, the applicability of the RSS and the determination of a proper value of p depend on the requirements and constraints of the application. In field reconstruction scenarios, the RSS can guarantee an average number k of nodes to be active in each sampling round by simply setting $p = k/N_{tot}$. As discussed previously in this chapter, however, for the purpose of field reconstruction the k selected sensors must also offer a high degree of coverage of the RoI. Therefore, the applicability of the RSS in this context is limited, unless a link between k and the expected level of sensing coverage (ESC) can be established.

Indeed, if the nodes are deployed at random over the RoI according to a known distribution, e.g. uniform or Poisson, it is possible to link the number of active nodes with the ESC their activation allows to achieve [40, 66, 79, 110, 111, 200]. For instance, when the nodes are deployed uniformly at random over the RoI an explicit analytical relation between k and ESC exists [40, 66], as we also detail in section 4.7.2 below. Thus, once the desired number of active nodes k is known, the RSS can come into play as a simple but effective sensor selection strategy.

Performing sensor selection using the RSS has several advantages. First, it requires an extremely limited control overhead, since it does not need coordination among neighboring nodes. Further, the random nature of the RSS implicitly provides for load balancing. Finally, the RSS is also robust against node failures or replacements, provided the number of nodes N_{tot} is sufficiently high. On the other hand, guaranteeing high levels of coverage using the RSS typically requires the activation of a number of nodes k that is much higher with respect to the figures achieved by specialized coverage preserving algorithms. In section 4.7.3, we address this inefficiency of the RSS in detail and suggest a method to mitigate it by letting sensor nodes autonomously determine the most appropriate value of their probability of activation p. To this end, we resort to the sensor ranking heuristics presented in section 4.5. Our experimental results, presented in section 4.7.4, show that this optimization allows to limit the number of active nodes k while still guaranteeing high levels of coverage. Before going into further details, however, we briefly summarize some useful theoretical results in the next section 4.7.2.

4.7.2. Coverage by Randomly Deployed Sensor Nodes

Under the assumption that N_{tot} sensor nodes are deployed over the RoI according to a known random process, it is possible to determine, at least asymptotically, with which probability they can offer complete coverage of the RoI [40, 66, 79, 110, 111, 200]. Clearly, this probability depends on the relation among the number of nodes, the form and extension of their sensing areas as well as on the form and extension of the RoI.

In 1947 Garwood analyzed this issue but with reference to a bombing problem [66]. In this context, he derives analytical expressions for the expected value and variance of the portion of area that remains uncovered after N_{tot} objects have "fallen" at random on a square or rectangular RoI. Both the cases in which the area covered by each of the N_{tot} objects is a circle or a rectangle are considered. Using Garwood's results, it is possible to derive an analytical expression for the ESC offered by k nodes selected at random among the N_{tot} available units. In particular, if the nodes have sensing range R_s and can cover a circular area centered at the node and having radius R_s, and the RoI is a square of side L, Choi and Das shown that the ESC can be expressed as [40]:

$$ESC = 1 - \left(\frac{L^2 + 4LR_s}{L^2 + 4LR_s + \pi R_s^2}\right)^k \tag{4.12}$$

Accordingly, the minimal number of nodes k that can guarantee the level of coverage ESC to be achieved (on average), is [40]:

$$k = \left\lceil \frac{\ln(1 - ESC)}{\ln\left(\frac{L^2+4LR_s}{L^2+4LR_s+\pi R_s^2}\right)} \right\rceil. \tag{4.13}$$

Garwood's results, and thus also Choi and Das', hold under the assumption that the nodes are deployed over an area A that contains, but does not coincide with, the RoI. In particular, for the equations reported above, the RoI is a square of side L, while A is a "*square with rounded corners*" with boundary at constant distance R_s outside the RoI [66], as schematically depicted in figure 4.13. This hypothesis significantly simplifies the analytical treatment of the problem, since it allows to prescind from the boundary effects. Indeed, at the boundary of a region the density of the nodes changes and this cannot be ignored when deriving equations like 4.12 and 4.13, as shown in [110, 111, 200]. In the following, we make use of equation 4.13 and thus assume the

deployment region to have the the form depicted in figure 4.13.

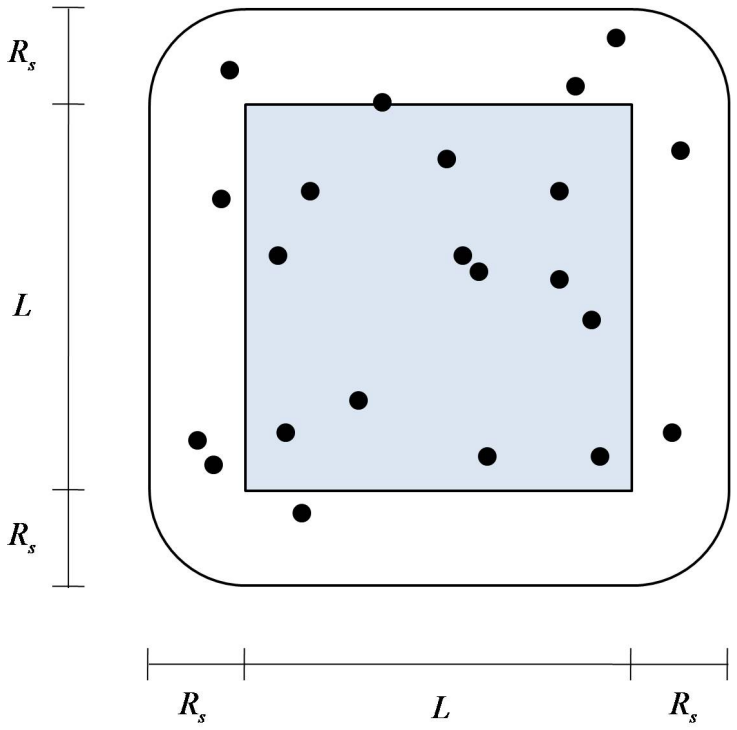


Figure 4.13.: Reference geometry for the derivation of equations 4.12 and 4.13.

4.7.3. Determination of the Probability of Activation

As mentioned above, guaranteeing a given ESC using the RSS requires the activation of an often unnecessarily high number of nodes k. For instance, for the experimental setting reported in figure 4.9, complete coverage of the RoI is achieved for all the 25 random configurations with fewer than 140 nodes (70% of the 200 nodes deployed in total). Instead, equation 4.13 indicates that, for the same setting, guaranteeing the coverage of 99% of the RoI using the RSS would require the unrealistic activation of $k = 245$ nodes.

Clearly, in most practical settings, the actual required value of k is typically much lower. In this context, the definition of practical heuristics to set the value of k, and thus p, are required. At the beginning of section 4.5.3, we noticed that the value of Ψ_i, as defined in equation 4.6, actually represents the probability that the node n_i must become active to guarantee coverage of its sensing neighborhood. Thus, the value of Ψ_i represents a suitable choice for the value of the probability of activation p_i of node n_i. Since with this choice the value of p_i is

adapted to the local topology, we refer to this strategy as the adaptive random sensor selection (ARS) scheme.

Our experimental results, presented in the following section 4.7.4, show that the ARS is effective in guaranteeing high level of coverage using a number of nodes k far smaller than what equation 4.13 would suggest. Furthermore, it offers a means to set the values of the probability of activation in a distributed manner, provided the nodes are aware of the required spatial resolution Δ_s and know the number and position of their neighbors. As for the latter requirement, we should recall that this information is usually available at the routing layer, and we thus assume that the ARS can access it without generating additional overhead.

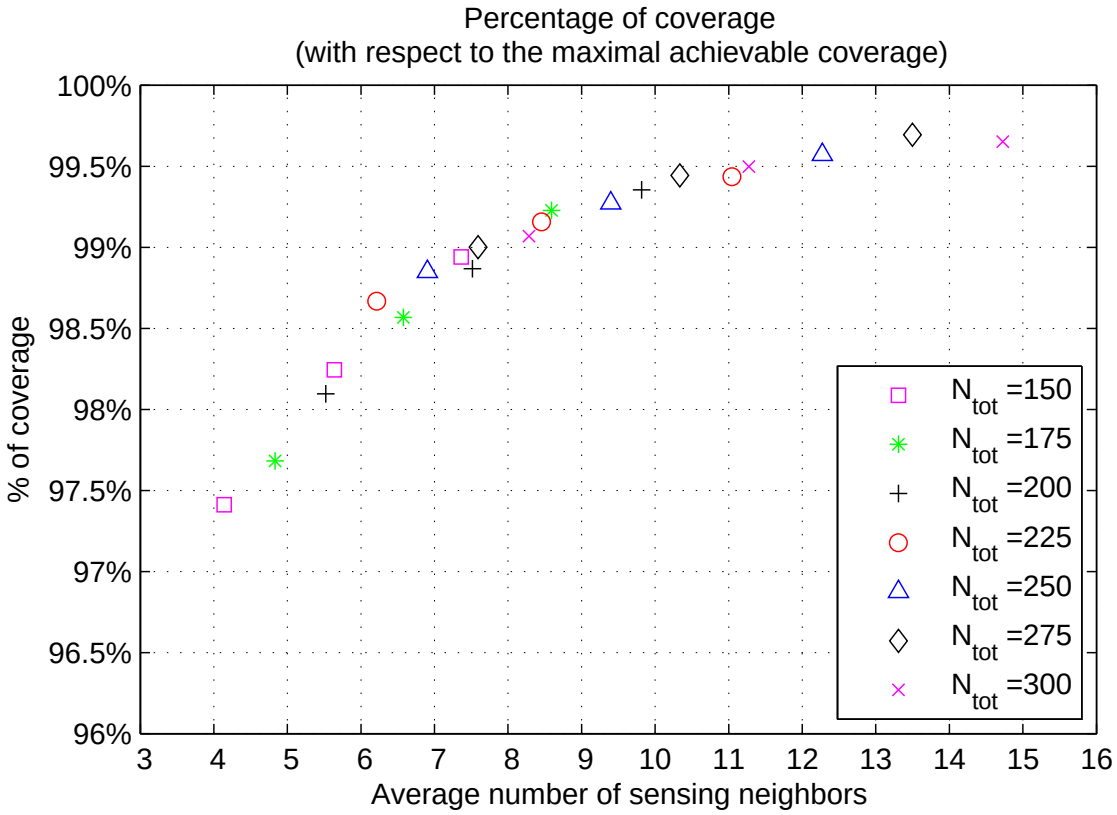


Figure 4.14.: Expected sensing coverage achievable using the ARS strategy, as the number of sensing neighbors increases. Results are in percentage with respect to the coverage achievable by the activation of all the N_{tot} available nodes. Experimental setting (2-dimensional case): $L_x = 100m$, $L_y = 100m$, $R_{tx} = 25m$, $Delta_s = 18.75, 21.875,$, and $25m$, $N_{tot} = 150, 175, 200, 225, 250, 275$, and 300.

4.7.4. Experimental Results

Figure 4.14 shows the ability of the ARS to provide for a high degree of coverage of the RoI, depicted as a function of the average number of sensing neighbors. Each data point in figure 4.14 is the average, over 25 random network configurations, of the level of coverage achieved by the ARS. For each configuration, the average level of coverage has been computed over 25 trials. As for the network parameters, we refer to the same setting defined in section 4.6.3 ($L_x = L_y = 100m$, $R_{tx} = 25m$). The average number of sensing neighbors is computed as $\pi R_s^2 N_{tot}/(L_x L_y)$. The total number of available nodes N_{tot} varies from 150 to 300, while the spatial resolution $\Delta_s = 2R_s$ varies from 75% to 100% of the value of R_{tx}.

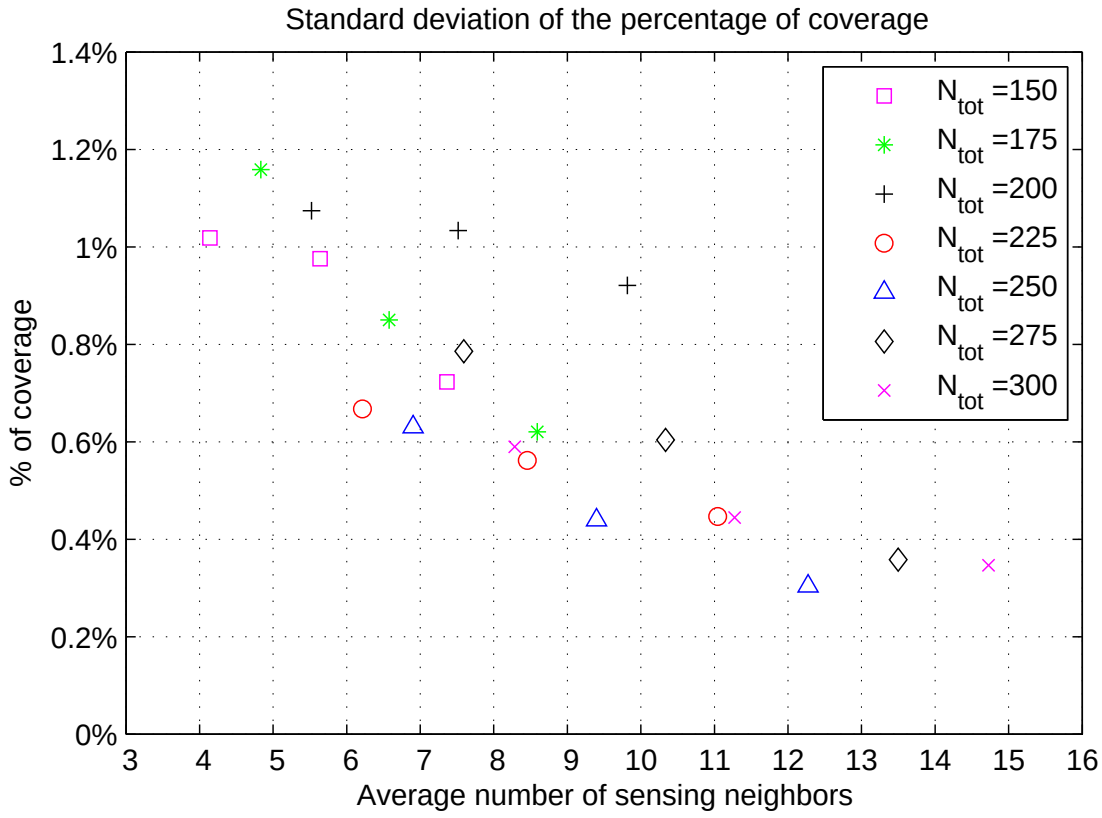


Figure 4.15.: Standard deviation of the expected sensing coverage achievable using the ARS strategy, as the number of sensing neighbors increases. Results are in percentage with respect to the coverage achievable by the activation of all the N_{tot} available nodes. Experimental setting (2-dimensional case): $L_x = 100m$, $L_y = 100m$, $R_{tx} = 25m$, $Delta_s = 18.75, 21.875,$, and $25m$, $N_{tot} = 150, 175, 200, 225, 250, 275,$ and 300.

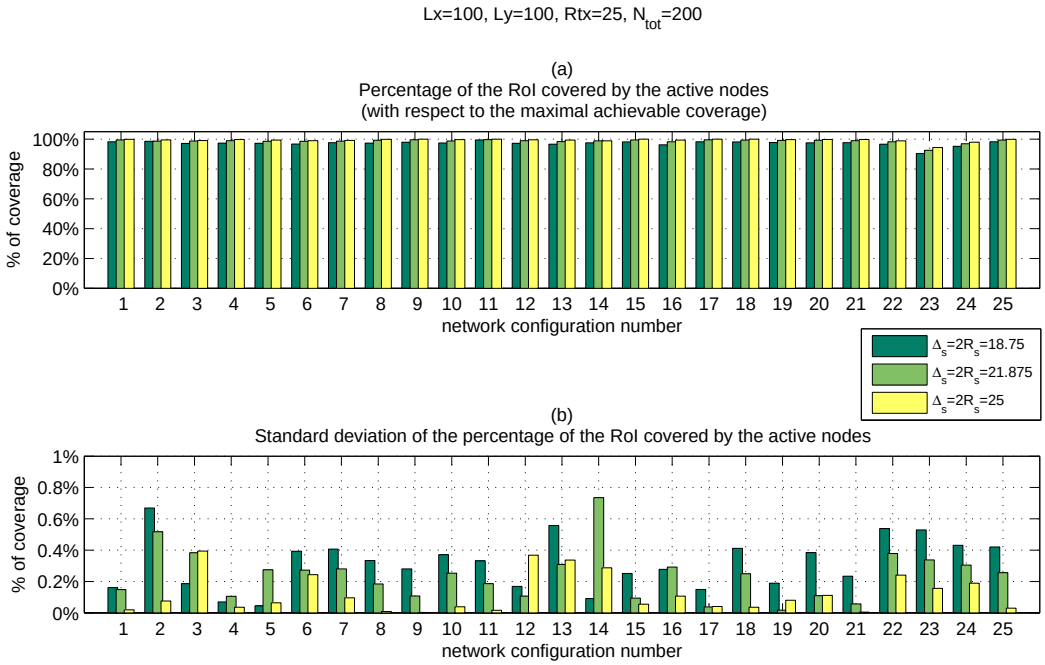


Figure 4.16.: Expected sensing coverage achievable using the ARS strategy (a), and corresponding standard deviation (b). Results are in percentage with respect to the coverage achievable by the activation of all the N_{tot} available nodes. Experimental setting (2-dimensional case): $L_x = 100m$, $L_y = 100m$, $R_{tx} = 25m$, $Delta_s = 18.75, 21.875$, and $25m$, $N_{tot} = 200$.

Figure 4.14 validates our claim that the ARS can provide for a high degree of coverage of the RoI. To show that these results are robust, we also report the corresponding values of the standard deviation in figure 4.15. As we can see, the standard deviation rarely exceeds 1%, and, as expected, its value shrinks as the average number of sensing neighbors increases.

Figure 4.16 shows the average level of coverage for all the 25 configurations, when $N_{tot} = 200$, along with the corresponding standard deviation (computed over 25 runs). As we can see, only in two cases (networks 23 and 24), the level of coverage achieved by the ARS is clearly below average. Figure 4.17 also show the corresponding average number of active nodes and its standard deviation. For all the three values of R_s considered here, the number of active nodes is between 60% and 80% of the total number of available nodes. Using equation 4.13, achieving the level of coverage of figure 4.16 would have required the activation of more than N_{tot} nodes.

Clearly, the superior performance of the ARS with respect to the RSS depends on the fact that the former is able to adapt to the local

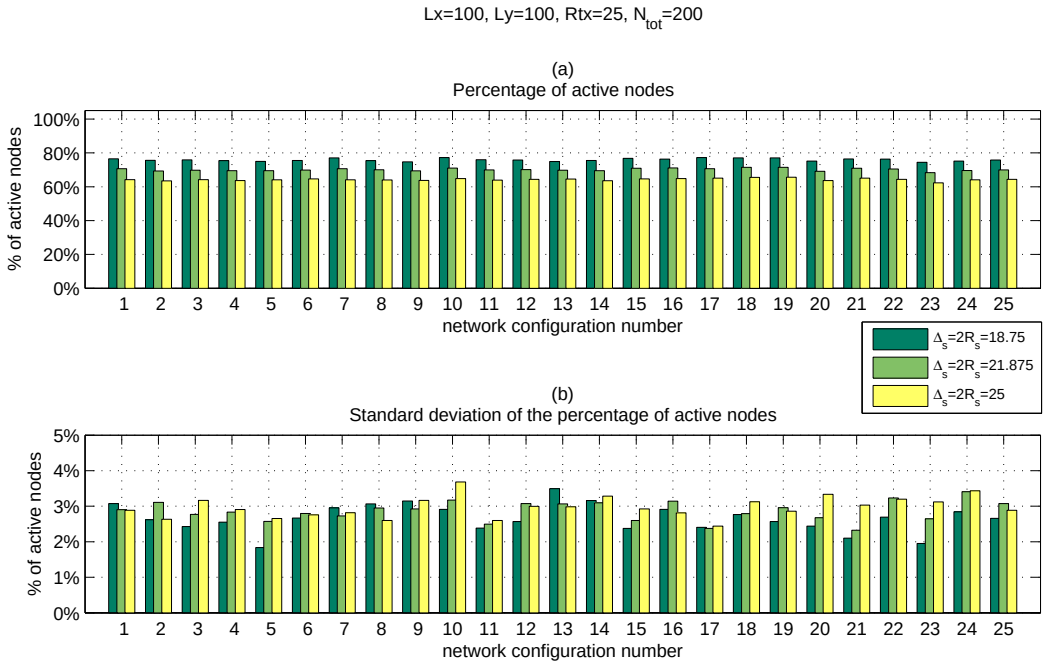


Figure 4.17.: Average number of active nodes using the ARS strategy (a), and corresponding standard deviation (b). Results are in percentage with respect to the total number of available nodes N_{tot}. Experimental setting (2-dimensional case): $L_x = 100m$, $L_y = 100m$, $R_{tx} = 25m$, $Delta_s = 18.75, 21.875$, and $25m$, $N_{tot} = 200$.

topology of the network. The ARS can thus select the proper value of the probability of activation p for each configuration, while the RSS uses a theoretically sound, but generic estimation. Although we also experimented with the use of other sensor ranking heuristics to set the values of p, we have found the method defined in section 4.5.2 to be the most reliable in terms of achievable coverage.

4.8. Integration of Sensor Selection and Routing

In order to report collected sensor readings to one or more sink nodes the sensor selection techniques discussed in this chapter rely on an underlying routing protocol to be available. The interplay between the sensor selection and routing layer is, however, very limited. In particular, we only assumed the routing table to be visible at the sensor selection layer, so that information about the presence and position of neighboring nodes can be retrieved.[7]

[7] Distributed routing protocols for WSNs typically require the nodes to periodically exchange routing *beacons* to build the routing tables. Depending on the specific protocol, the information included in these messages and the rate at which they are sent may vary [104, chapter 11]. In

In principle, decoupling the sensor selection and routing logic is desirable, since it allows for modularity and portability. On the other hand, it may also result in an inefficient distribution of the energy consumption within the network. Indeed, if no coordination exists between the two layers, nodes that are frequently selected for sensing may also be required to regularly operate as data routers. This can make such nodes deplete their batteries sooner than other, less busy peers. Properly balancing the participation in sensing and routing of single nodes may allow to increase their lifetime, and, thus, that of the whole network. Achieving this balance requires applying appropriate cross-layer optimization techniques [104, chapter 13]. In particular, previous work has shown the effectiveness of a combined approach to sensor selection and routing in increasing network lifetime [147, 150, 151, 168]. Drawing upon this work, we are interested in investigating the possibility to use the sensor ranking heuristics presented in section 4.5 to better balance the participation of a node in routing activities. In particular, we performed a preliminary, quantitative evaluation using the Collection Tree Protocol (CTP) as a reference routing mechanism.[8]

CTP is a generic routing protocol recently developed within the TinyOS community that can reliably report data to one or more sink nodes [58, 68]. CTP constructs one or more forwarding trees having each a sink as their root. To this end, each node must select one of its neighbors as a parent in the tree according to the so-called *Expected Transmissions* (ETX) metric. The ETX is a complex link quality indicator that takes into account several factors, like the fraction of lost messages over the total sent, or the decoding error at the physical layer. The ETX of a node n_i basically estimates the number of transmissions required to deliver a data packet to (one of) the sink(s), if it is routed through the node n_i itself. Among the set of its neighbors, a sensor node selects as its parent the one with the lowest ETX. Clearly, if a node has high ETX and sensor rank, it will be frequently selected both as sensor and data router, and will thus rapidly deplete its batteries. To cope with this problem, nodes with high sensor rank should be possibly preserved from acting as data routers. In particular, these nodes could "downgrade" their ETX so as to reduce their probability to be chosen as data routers by other nodes. In principle, this modified ETX, which

this context, we assume that the position of the nodes is included in these beacons and, thus, in the routing table.

[8] For the interested reader, we describe this protocol in detail in appendix B.

Table 4.1.: Tested functions for including the sensor rank SR in the computation of the ETX metric.

Experiment	ETX
1	standard ETX
2	$ETX_{mod} = ETX \cdot (1 + \frac{SR}{4})$
3	$ETX_{mod} = ETX \cdot (1 + SR)$
4	$ETX_{mod} = ETX + \alpha SR$, $\alpha = 2$

we call ETX_{mod}, can be computed as an arbitrary function of the real ETX and the sensor rank SR, thus as $ETX_{mod} = f(ETX, SR)$. To understand the feasibility of this approach and test suitable weighting functions, we implemented CTP on the Castalia WSN simulator, which we introduced in section 2.2.3, and performed a preliminary experimental study. The implementation of CTP in Castalia as well as the considerations reported in this section are the outcome of a joint effort with Ugo Colesanti, of the "Sapienza" University of Rome.

In particular, we run a series of experiments using the following setting. We consider a network with N_{tot} nodes deployed uniformly at random over a square area of side lengths L, and all having the same transmission and sensor range R_{tx} and R_s, respectively. We let the nodes wake up at regular time intervals and decide upon their participation in sensing using the ARS strategy introduced in section 4.7. Concurrently, the nodes participate in the construction of the CTP routing tree using either the ETX metric or its modified version ETX_{mod}. We consider three different empirical functions to compute the ETX_{mod}, as summarized in table 4.1. We then generate 50 different network configurations and run 50 trials (i.e., sampling rounds) for each network and each different "type" of the ETX. For each trial, we measure the number of data packets and routing beacons transmitted and received by each node. By examining the load in terms of sent and received packets with and without using the modified version of the ETX, we can measure the effectiveness of the ETX_{mod} in providing for better load balancing. In particular, we compute the difference between the total transmitted and received packets with and without the ETX_{mod}, in percentage with respect to the performance obtained without ETX_{mod}. We refer to this performance measure as the *differential total transmitted and received packets*.

Figure 4.18 shows, for a specific network configuration, the average

differential total transmitted and received packets as a function of the sensor rank and for all the three considered expressions of ETX_{mod}, when the parameters N_{tot}, L, R_{tx}, and R_s take the values 100, $250m$, $50m$, and $25m$, respectively. As we can see, the use of ETX_{mod} in spite of the plain ETX makes nodes with lower sensor rank to route a significantly higher number of packets with respect to the case in which the ETX is used. On the other hand, the vast majority of nodes with a high sensor rank can reduce their communication load, although there are some exceptions. For this specific network configuration, the influence of the function used to compute the ETX_{mod} is limited, but, on average the formula $ETX_{mod} = ETX + \alpha SR$ provided the better performance on most configurations. In general, however, we observed a high variability in performance depending on the specific network configuration. For instance, the average total number of transmitted data packets and beacons may increase or decrease as a consequence of the introduction of the ETX_{mod}. Furthermore, although in most configurations the majority of the high rank nodes could reduce their communication load, in some cases we also observed some of these nodes significantly increasing their messaging overhead.

In the light of these considerations, we cannot provide a decisive characterization of the performance of the proposed interaction between CTP and our ARS strategy. However, we believe the investigation of this interaction constitutes a promising direction for further research.

4.9. Summary

In this chapter, we presented our approach to the spatial sensor selection problem in the specific context of field reconstruction applications. In particular, we first discussed the feasibility of the ACT reconstruction algorithm to be used in the context of WSNs. We then showed that, using this algorithm, the problem of selecting sampling geometries that can allow for good reconstruction performance can be reduced to a coverage problem. We thus focused on a specific, state-of-the-art coverage preserving algorithm, known as CCP, and proposed to leverage it to provide for adequate short-lived sampling geometries. In this context, we introduced a set of heuristics that allowed to improve the performance of CTP by reducing the communication overhead required to select a short-lived coverage preserving sampling geometry. Further, we showed how the considered heuristics can also be leveraged to im-

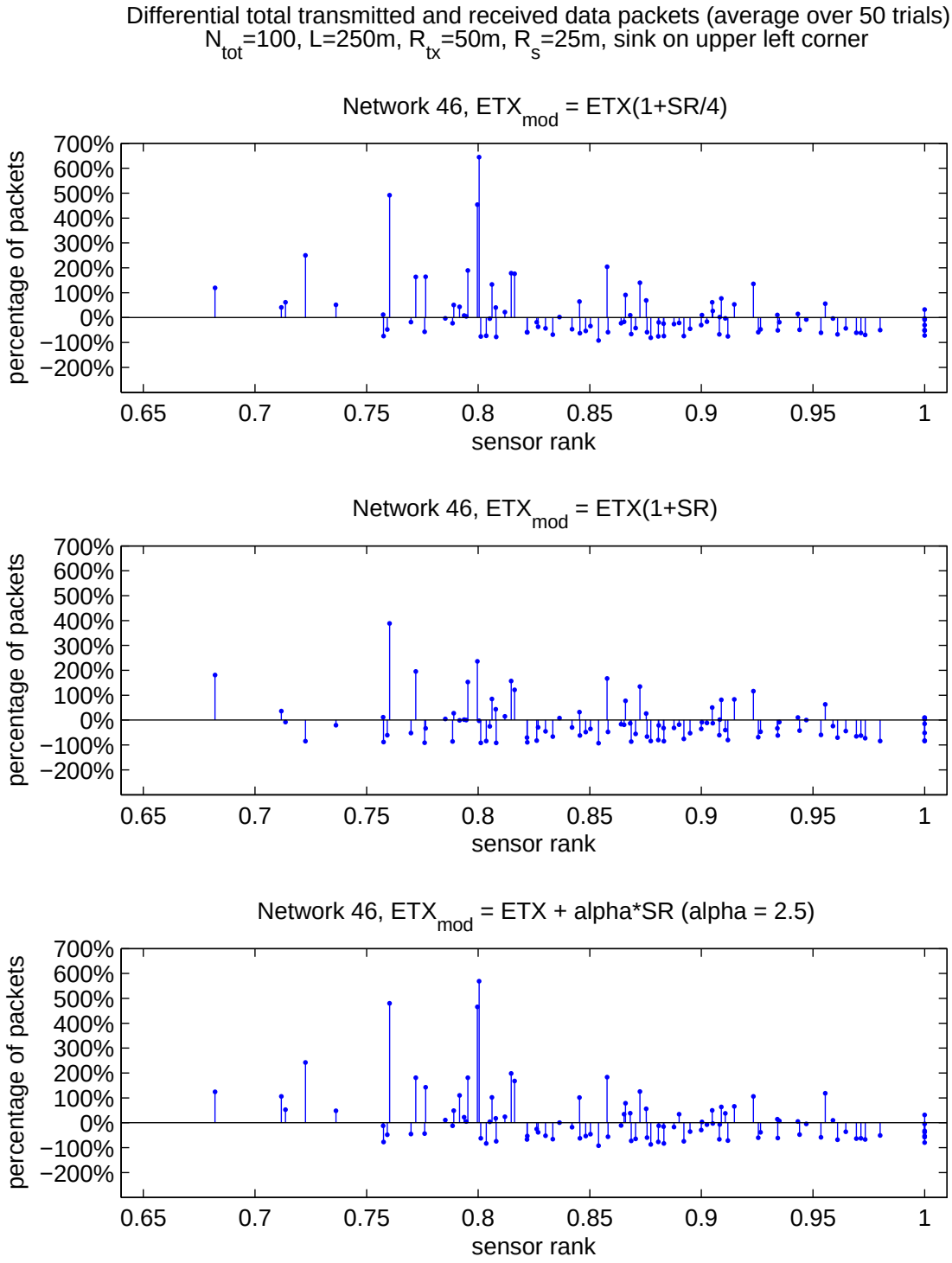


Figure 4.18.: Differential total transmitted an received packets for network 46, averaged over 50 trials, as a function of the sensor rank and for all the three considered expressions of ETX_{mod}. Experimental setting (2-dimensional case): $L_x = L_y = L = 250m$, $R_{tx} = 50m$, $R_s = 25m$, and $N_{tot} = 100$.

prove the performance of a random sensor selection strategy. Finally, we provided some qualitative and quantitative remarks on the interplay between spatial sensor selection and routing.

5. Environmental Noise Monitoring – An Application Scenario

In the previous chapters, we have addressed the sensor selection problem in typical, generic application scenarios for wireless sensor networks (WSNs), like environmental monitoring and sensor field reconstruction. In this chapter, we investigate a concrete application scenario in order to assess its suitability for our sensor selection strategies, namely the monitoring of noise pollution levels in urban environments.

Noise pollution is "*one of the main environmental problems in Europe*" [53, page 1a]. In order to develop, enforce, and validate policies aiming at the abatement of noise pollution across European member states, the possibility to accurately assess its actual levels is considered instrumental [53, 187]. In this context, WSNs can be used to perform noise measurements with an accuracy that current assessment procedures cannot achieve. For this application scenario, we initially assumed both our prediction-based sensor selection strategy (cf. chapter 3) as well as our ARS strategy (cf. chapter 4) to be readily applicable. However, a closer analysis (see section 5.3) showed that due to the particular requirements of the domain, the applicability of our algorithms is somewhat limited.

In fact, a closer examination of the problem area revealed that one of the major challenges towards the implementation of a noise monitoring application using WSNs actually consists in the selection of a suitable sensing platform. Several issues (e.g., the lack of adequate hardware calibration procedures and the fidelity of the audio sensing) hinder the use of commonly available sensor nodes as reliable noise pollution sensors [165]. Equally unsuitable seems to be the use of mobile phones as ubiquitous mobile noise meters, e.g., as suggested in [38, 177]. To overcome these limitations, it is instead possible to leverage commercially available noise level meters. Once properly interfaced to a generic WSN platform such as the Tmote Sky, these devices can operate as external

sensors and provide for reliable noise level measurements.

In the following section 5.1, we report the definition of environmental noise and discuss the recent efforts promoted by the European Commission to foster a comprehensive assessment of its levels in urban environments. In this context, we describe the potential impact of WSNs as a flexible tool to perform fine-grained measurements of noise levels. We then provide an overview on related work in section 5.2, followed by an analysis of the application requirements in section 5.3. In the same section we also discuss the applicability of the sensor selection strategies presented in chapters 3 and 4 to this specific application scenario. We then report our experiences in using both sensor nodes and mobile phones as noise pollution sensors in sections 5.4 and 5.5, respectively. Finally, section 5.6 summarizes and concludes the chapter.

The results reported in this chapter are the outcome of a collaboration between the author and Andrea Vitaletti, of the "Sapienza" University of Rome, Italy, as well as a joint work with Benedikt Ostermaier and Robert Adelmann of the ETH Zurich, Switzerland. Most of the contributions described below have been published in [57, 164, 165, 167].

5.1. Motivation and Background

Noise is defined as sound that is "*loud, unpleasant or unexpected*" [53]. Typical noise sources are road and railways traffic, airplanes, industrial equipment, and human activities such as discussions or music playing. The compound effect of these noise sources is referred to as *environmental noise* or *noise pollution* and is measured using so-called *noise indicators*. These indicators are in turn determined as average loudness levels over specific periods of time, like a day, night, week, or even years.

In 1996, the European Community estimated that about 80 millions of its citizens (20% of the total at that time) were exposed to unacceptable levels of environmental noise, and that another 170 million suffered serious annoyance from high noise pollution during daytime [53]. Among the many negative consequences of noise pollution on human health, the World Health Organization (WHO) lists hearing impairment, disturbance of rest and sleep, and increased blood pressure and heart rate [18]. The high number of citizens exposed to environmental noise, and the related health risks (and thus costs), prompted the European Commission to declare the abatement of noise pollution as

one of the main goals of its environmental protection policy. Directive 2002/49/EC of the European Parliament (in the following simply called "the Directive") has since made the avoidance, prevention, and reduction of environmental noise a prime issue in European environmental protection policy. In particular, the Directive requires member states to determine the exposure of its citizens to environmental noise, and to ensure that information on noise pollution and its effects is made available to the public [187].

According to the Directive, member states are required to provide an accurate mapping of environmental noise exposure for all agglomerations with more than 250 000 inhabitants, as well as along major roads, railways, and airports [187, Art.7]. Adequate local action plans for the abatement of noise can then be elaborated based upon such noise-mapping results [187, Art.1].

While current noise maps are mostly based on sparse data and ad-hoc noise propagation models, the Commission has stressed that *"every effort should be made to obtain accurate real data on noise sources,"* [54, page 6]. The demand for accurate data about noise exposure levels is likely to increase significantly, as this statement makes its way into mandatory regulation. In this context, the use of WSNs could help in satisfying this rising demand for fine-grained noise data. In particular, WSNs could provide noise measurements with an accuracy and cost-efficiency that current noise assessment procedures cannot afford. Indeed, today's noise measurements are mainly carried out manually by designated officers that collect data in a location of interest for successive analysis and storage, using a sound level meter.[1] This collection method using expensive equipment does not scale well as the demand for higher granularity of noise measurements in both time and space increases. Instead, a network of wireless sensor nodes deployed over the area of interest could continuously collect noise pollution data over days or weeks, and autonomously report it to a central server through the onboard radio of the nodes. Human intervention would then be required only to install and subsequently remove the sensing devices. Moreover, since sensor nodes are typically equipped with several different sensors, they can label the collected noise data with additional information like, e.g., the temperature and humidity values registered as the noise mea-

[1] The author is indebted to Hans Huber and Fridolin Keller of the department for environmental noise protection of the city of Zurich, Switzerland, who described the current practice in a personal in-depth interview in March 2007 [88].

surements were collected. This information should indeed be provided for any properly collected set of noise exposure data [88].

WSNs could also provide for the validation of noise levels obtained using computational models. Noise data is typically stored in a land register and used, together with additional information about existing noise sources, to feed computational models that provide extrapolated noise exposure levels for those areas for which real data is unavailable. Even if this assessment procedure is still compliant with European regulations, computational models often fail to provide accurate estimations of the real noise pollution levels [88]. While the free propagation properties of noise generated from typical noise sources are well understood, shadowing and reflection effects hinder accurate estimation of noise levels in complex urban settings. For instance, estimated noise levels on internal buildings façades (e.g., facing a courtyard) are typically unreliable, and this inaccuracy may become critical if noise exposure data is used to drive decisions about construction planning or to elaborate local noise abatement policies [88]. The accuracy of estimated noise levels could thus be verified and improved by installing a WSN at potentially sensitive spots (like, e.g., schools or hospitals) or at those locations for which computational models are likely to provide inaccurate estimations. Indeed, the European Commission points out that real noise measurements can be used to "*validate noise maps at selected sites*", "*boost public confidence in these maps*", and "*show the real effects of action plans once they are implemented*" [54, page 10]. To perform such measurements, noise assessment points must possibly be closely spaced, and data should be collected simultaneously at all assessment points. While this distributed sensing setup is cumbersome and costly to realize with the current measurement procedures, it is a "natural" setup for WSNs.

Last but not least, the use of WSNs may also bring significant improvements in the assessment of noise pollution due to vehicular traffic on urban roads. The current procedure requires estimating, for several different vehicle classes, the average number of units passing-by at daytime, evening and night and the average noise level during each vehicle pass-by [24, 93]. This estimation is either performed through computation, with the drawbacks and problems outlined above, or it is performed manually, i.e., by a designated officer standing nearby the road and annotating the type and number of vehicles passing-by [88]. WSNs have already proven their ability to detect and classify vehicles [7] and

could therefore be used in this context to automate the vehicle counting procedure and, at the same time, record the corresponding noise levels.

Summarizing the considerations reported above, the use of WSNs in the context of noise monitoring could bring significant advantages in at least three specific application scenarios. First, to collect fine-grained data to be included in noise maps. Second, to validate the values of noise pollution levels determined using computational models. Third, to count the number of transiting vehicles in road traffic monitoring. To the best of our knowledge, we have been the first within the WSN research community to address these specific application scenarios within the broader context of environmental noise monitoring [167]. WSNs could be highly suitable to provide for noise mapping data at selected locations, like building façades, internal courtyards, or public parks. While the costs and efforts required to deploy a dense WSN to measure noise levels over an entire neighborhood or city still appear prohibitive, small targeted deployments or multi-purpose installations (i.e., to also measure other environmental effects) might soon make early inroads. Also, the recent rise of urban-scale sensing infrastructures [38,138] could provide another avenue for rendering comprehensive urban noise mappings feasible in the foreseeable future.

In our initial investigations, however, we focus on scenarios in which a WSN is deployed over a geographically limited region, like a building façade. In this setting, the WSN can collect noise level data with the modalities discussed in section 5.3 and report its measurements to a base station for permanent storage and later analysis. The collected noise data can then be used to feed computational models and produce comprehensive noise maps, or to validate previously computed synthetic data. We summarize the main requirements of this reference application scenario, and provide practical guidelines for its implementation on WSNs platforms, in sections 5.3 and 5.4. In section 5.3, we will also discuss the applicability of the sensor selection strategies described in chapters 3 and 4 to the specific environmental monitoring scenario under consideration.

Before going into further details, however, we discuss related work in the following section 5.2. Besides considering related efforts in the context of "classical" WSNs, we also report on several recent approaches that aimed at capturing noise levels using mobile phones. Due to the rising interest in these platforms, we also evaluated their performance as noise pollution sensors, as we detail in section 5.5.

5.2. Related Work

Several authors within the WSNs research community have mentioned noise monitoring as a potential application scenario for WSNs [52, 208]. However, they did not provide a closer description of the specific requirements of the application nor investigated practical issues concerning its practical implementation. More recently, the *BikeNet* project [51] showed how average noise levels can be used to influence daily decisions like the choice of the cycling route to work. The prototype developed for the *BikeNet* project uses the Tmote invent platform to derive estimations of the actual noise levels in the immediate neighborhood of a cyclist. However, as we reported in [165] and discuss in section 5.4.2, using the Tmote invent platform to measure noise levels may cause the measurements to be significantly inaccurate.

There exists also a considerable body of literature dealing with applications for WSNs that exploit acoustic measurements, e.g., for target or event detection and classification (e.g., shooters, birds or volcanic eruptions), acoustic-based localization, or communication [47, 73, 191, 201]. For these applications, however, specific features of the acoustic signal, like its frequency spectrum (e.g., for birds classification) or the relative loudness or time shifting between two signals (for detection and localization) are of interest, and not the absolute loudness, like for the assessment of noise levels.

Recent efforts also investigated the possibility of using the microphones of commonly available mobile phones as ubiquitous acoustic sensors [38, 102, 130, 133]. For instance, the MobGeoSen system [102] used built-in microphones and other sensors attached to mobile phones to collect pollutant levels in an urban environment. Examples of noise levels, expressed in dB, are presented, but the accuracy of this data as well as its coherence with measurements taken from nearby phones is not discussed. As we show in section 5.5, however, measurements collected with mobile phones may be highly inaccurate.

Within Microsoft's Nericell project [133], audio recordings from the built-in microphone of a smartphone constitutes the input of a "honk detection" algorithm, which in turn feeds an estimator of the current traffic conditions. The authors investigate the influence of background noise and the sensitivity of the microphones on the performance of the honk-detector, but their approach does not support the assessment of the absolute of the actual noise levels. Similarly, the *CenceMe* system

[130] uses simple Python scripts to capture audio signals on Nokia $N80$ and $N95$ mobile phones, but the data is processed on the mobile phone to determine whether it contains voice or just background noise, and not to assess noise levels.

Other authors investigate the challenges and possibilities related to the use of mobile phones as "complex" sensors. For instance, Misra et al. [131], present several examples of how the microphone of mobile phones can be used for music applications. Furthermore, they underline that, being the development of mobile operating systems still ongoing, writing applications for mobile platforms that rely on such systems may be cumbersome and time consuming. On the other hand, the recent spread of flexible platforms like Apple's iPhone[2], or Google's Gphone/Android[3] fostered the appearance of a high number of readily available sensing applications. For instance, as of January 2009 at least 10 applications for capturing noise levels using the iPhone were available on the *App Store*[4], Apple's portal for the publication and selling of customized applications for the iPhone. This shows a growing public interest in the issue of noise pollution, but the actual accuracy of the measurements collected using the mentioned applications remains questionable, as we show in section 5.5.

Finally, we would like to mention that the European Commission also funded a number of projects to foster the enforcement and improvement of regulations concerning the monitoring of noise pollution levels [35, 36]. Many of these efforts aim at the definition or improvement of common noise assessment procedures and data management infrastructures across European member states. As for the actual technology used to perform noise measurements, the Commission mainly relies on established international standards [91–93].

5.3. Application Requirements and Applicability of Sensor Selection

As discussed at the end of section 5.1, we consider a scenario in which a WSN is deployed in a limited geographical area like the façade or internal courtyard of a building. In this setting, the network can compute

[2] `www.apple.com/iphone`

[3] `www.android.com`

[4] `www.apple.com/iphone/apps-for-iphone`. See, e.g., the Widenoise project: `www.widetag.com/widenoise`.

appropriate noise indicators and report them to a central server, where they are processed and stored for later utilization. In the following, we first briefly describe how to compute these indicators according to the international standards and guidelines indicated by the European Commission [54, 93, 187]. We then summarize the requirements that a system for environmental noise monitoring based on WSNs should be able to comply with. By distilling these requirements, we also contribute several practical hints and references for developers interested in implementing a WSN application for environmental noise monitoring. On the basis of these considerations, we also discuss the applicability of the sensor selection strategies proposed in chapters 3 and 4 to this particular application scenario.

5.3.1. Computation of Noise Indicators

Noise pollution levels can be specified using different indicators, depending on the particular purpose of the noise measurement. For the preparation of noise maps, the European Commission indicated the *equivalent continuous sound pressure level* $L_{Aeq,T}$ as the indicator of choice [187, Annex I]. The $L_{Aeq,T}$, which we also refer to as *equivalent noise level*, is defined as [93, page 3]:

$$L_{Aeq,T} = 10\log_{10}(\frac{1}{T}\int_0^T \frac{p_A(t)^2}{p_0^2}dt), \tag{5.1}$$

where $p_A(t)$ is the A-weighted instantaneous sound pressure produced by an acoustic wave, and p_0 is a standard reference value corresponding to the minimal (human-) audible acoustic signal (i.e., $20\mu Pa$). A-weighting is a frequency filtering technique that simulates the frequency response of the human ear [92, page 2-3], [24]. A-weighted noise indicators are accordingly indicated in A-weighted decibels or dB(A). The period T over which the $L_{Aeq,T}$ indicator is computed may vary depending on the specific noise source or area of interest, and may be "*part of a day, the full day, or a full week*" [92, page 4], or any other value properly defined by the competent authority. Noise levels meters able to measure the $L_{Aeq,T}$ indicator over a settable interval T are known as *integrating-averaging sound level meters* [91, page 9]. Most commercially available integrating-averaging sound level meters offer the possibility to store the so-called *short* $L_{Aeq,T}$ samples. These values represent the samples of $L_{Aeq,T}$ taken at short time intervals, in the

order of few seconds or less.[5] Their collection allows to visualize the noise level over time and build a database upon which the values of $L_{Aeq,T}$ over arbitrary intervals T can be computed. We would also like to note that the $L_{Aeq,T}$ indicator actually represents the level of a constant noise source over the time interval T that has the same acoustic energy as the actual varying sound over the same interval.

The equivalent noise level $L_{Aeq,T}$ defined above drives the computation of those specific noise indicators that are used for the preparation of noise maps. In particular, European member states must provide noise pollution data (at least) in terms of the $L_{Aeq,T}$ averaged over the night only and over the whole day [187, Annex I]. The directive defines the duration of the *day* (d), *evening* (e) and *night* (n) periods, to be 12, 4, and 8 hours, respectively, and the default start of the day to be at 7:00am (local time at the measurement location). The indicators for the preparation of noise maps are thus accordingly indicated as L_{night} and L_{den}. Their values "*should reflect the average calculated over the continuous period of twelve months of a relevant calendar year*" [54, page 12]. This average can be computed using adequate forecasting techniques, possibly extrapolating from short-term real measurements [54, page 12].

A WSN set up for environmental noise monitoring could provide for these short-term measurements over several days or weeks, and thus enable a more reliable computation of the yearly averages.

5.3.2. Application Requirements

In our target application scenario, a WSN is set up to measure the equivalent noise level defined by equation 5.1 at several locations over an area of interest. In the following, we list the requirements involved in implementing this application. To this end, we resort to the requirements taxonomy defined in [160], and provide our considerations according to the thereby defined categories. We distilled the here discussed application requirements from the pertinent European directive, studies, and international standards [91–93, 187]. Table 5.1 provides a brief overview of the results of our analysis.

[5] Many sound level meters with data logging capability offer the possibility to store short $L_{Aeq,T}$ values at a rate variable between 1/16 and 16 seconds. See for instance the CR:703B and CR:704B devices by Cirrus Research [41].

Dimension	**Class**
Deployment	Manual, one-time
Network size	Tens of nodes
Coverage	Dense
Mobility	Immobile
Cost, Size, Resources, and Energy	Bricks
Heterogeneity	Homogeneous
Communication modality	Radio
Infrastructure	Single base station
Network topology	Possibly multi-hop
Connectivity	Intermitted/connected
Network lifetime	Weeks
Other QoS requirements	Calibration

Table 5.1.: Application requirements of our reference environmental noise monitoring scenario. The requirements categories and classes have been defined according to the taxonomy proposed in [160].

Deployment, Network Size and Coverage. Sensor nodes are manually placed at specific locations over the area of interest. In particular, the positions of the measuring devices, or assessment points, must follow rules defined in [187, Annex I], [93, pages 7-8], and [54]. Near to buildings, for instance, the assessment points must be 4.0 ± 0.2m above the ground and at the most exposed façade. If necessary, other heights may be used but they shall never be less than 1.5m above the ground, and results should be corrected in accordance with an equivalent height of 4m [187, Annex I]. Further, in [54, page 44], it is suggested that "*a spacing of* 3 *meters between calculation points around the façade is likely to be appropriate*". Covering a façade of 20x15m would thus require the deployment of about 30 nodes. In general, we believe a few tens of nodes to be an appropriate estimate of the network size required to provide accurate noise mapping data and model validation setups.

Cost, Size, Resources, and Energy. The network must be able to measure the required values of $L_{Aeq,T}$, as discussed in section 5.3.1. To this end, sensor nodes must be equipped with an adequate noise sensing unit, like the integrating-averaging sound level meters mentioned above. According to international standards [91, 93], two certified classes of noise level meters, class-1 and class-2, may be used for the purpose of noise mapping. Being less accurate, devices belonging to the lat-

ter category are usually also much cheaper,[6] and should thus be used whenever possible. The form factor of typical noise level meters can be several times that of the Tmote Sky sensor node. Smaller devices are available, but often less accurate and/or harder to interface with commonly available sensor nodes.

Due to the need of letting the microphone and acoustic signal processing circuitry active for prolonged periods of time, the noise measurement unit may drain the batteries of a sensor node in a few days.[7] Therefore, the node should be able to easily switch the noise sensor on and off, so as to keep it active only when strictly necessary. For instance, if the network is set up to estimate the value of $L_{Aeq,T}$ over the evening period, the sensing unit must be active for only 4 hours a day and could be switched off during daytime and night.

Heterogeneity and Mobility. The sensing devices used to measure noise levels should be identical from both a hardware and software point of view. This can ensure a coherent comparison of measurements taken by different sensors. Furthermore, sensor nodes should not change their location after the initial deployment.

Communication Modality. To ease automated data collection, the sensing devices should be able to communicate wireless with a data collector. In the considered deployment scenarios optical and acoustic communication are prone to failures, due to, e.g., interferences with daylight or noise in the environment. Therefore, the preferred communication modality is radio.

We would like to point out at this stage that for the purpose of noise mapping and model validation the collected noise levels must not reach the data sink in real-time, and can therefore also be logged on the node for later reporting. In other words, there is no strict requirement on data latency. For instance, if the network is required to collect short $L_{Aeq,T}$ levels (at, e.g., 1 Hz rate), consecutive readings can be reported in a single packet, so as to save communication and,

[6] Depending on their accuracy and set of functionalities, noise level meters may cost between hundreds and several thousands of dollars. For instance, one unit of the Extech 407740 class-2 noise level meter we make use of in our experiments costs, as of August 2009, about $250.

[7] Unfortunately, we do not have generic figures regarding the power consumption of a noise level meter, since it also depends on the specific device and operation modalities. The noise level meter we used in our experiments, the Extech 407740 class-2 noise level meter, runs out of power in about 4 days, if operated continuously out of its 9V battery.

thus, energy. However, safely storing data on a sensor node is still nearly as expensive as data communication.[8] Therefore, it may still be convenient to report data immediately upon collection instead of storing it locally, especially if a direct link to the data sink exists. In this context, prediction-based data collection techniques, as those discussed in chapter 3, may come into play to reduce the number of transmissions, as we detail in 5.3.3 below.

Infrastructure and Network Topology. For the considered deployment scenarios, the presence of a powerful base station able to collect and report data to a remote database is assumed. Depending on the extension of the network, multi-hop communication may be required. In most cases, however, we expect a direct communication link between the nodes and the base station to be available.

Connectivity. Since latency is not a critical parameter for the application, intermittent connectivity is tolerated. However, since the network is static and dense, its is also likely to be connected.

Lifetime. The lifetime of the network is determined by the time needed to compute estimations of noise indicators with a desired accuracy. According to European directives, estimation of noise levels are considered sufficiently accurate if "*dividing them into crisp (discrete) 5 dB(A) wide sets is an appropriate process*" [54, page 53]. For a single sensor node, we can reasonably assume its measurements to be sufficiently accurate if the corresponding standard deviation is $\leq$ 2.5 dBA. However, the specific conditions for considering the measurements completed must be defined in a case-by-case manner.[9] With the current technology, however, the factor that mostly influences the lifetime of the network is the extension of the time interval over which the sensor must continuously collect data.

Other QoS Requirements. When collecting absolute noise levels, accurate calibration of the measuring devices is essential. Most commer-

[8] For instance, on the Tmote Sky sensor node, writing data on the flash memory drains 20 mA of current, thus more than radio transmission, which drains 17.4 mA (see also table 2.1). However, power efficient alternatives exist [128] and could be instrumental in improving the data storing efficiency of sensor nodes.

[9] For the specific case of road traffic noise, useful practical considerations regarding the variability of noise levels are reported in [4].

cially available noise level meters can be purchased together with an adequate, certified calibrator. Using this instrument, noise level sensing devices can be opportunely calibrated prior to deployment. However, if the measurement session takes place over a long period of time, calibration must be performed at least once a day [93, page 3]. The need for frequent re-calibrations actually represents the highest burden towards the practical deployment of a WSN for noise monitoring. Indeed, performing acoustic calibration (done using a standard acoustic signal and accordingly regulating the responses of the devices), would require to directly access the sensor nodes after deployment. On the other hand, electrical calibration (done using electrical, instead of acoustic, reference signals), would require putting additional hardware and logic on the nodes, thus significantly increasing their cost, power consumption, and form factor.

5.3.3. Applicability of Sensor Selection

On the basis of the above reported analysis of the application requirements, we can now discuss the applicability of our previously presented sensor selection strategies. In particular, we consider two specific scenarios in which prediction-based monitoring techniques (cf. chapter 3) or our ARS spatial selection strategy (cf. chapter 4) could be applied.

Let us assume that a network is set up to measure the values of the $L_{Aeq,T}$ over relatively short periods of time, e.g., every 5 minutes [93, page 9]. Depending on the time of the day at which the measurements take place and the corresponding location, successive samples of the $L_{Aeq,T}$ may be fairly stable (e.g., at night in a quiet neighborhood), or vary significantly (e.g., at daytime in a public park). If data is to be reported immediately upon collection, prediction-based techniques like those discussed in chapter 3 may come into play. In particular, if the difference between the current reading and the previously reported sample is within a tolerated accuracy, then the communication of the current sample could be suppressed, so as to save communication. Immediate reporting may be appropriate if safely storing data on the nodes is expensive (in terms of energy consumption), or if the collected data is also used to feed applications that estimate other environmental parameters, e.g., the current traffic level or the "busyness" of a place [38,177]. However, the power consumption due to the continuous operation of the noise level meter is likely to dominate the power

budget of the node. Thus, the savings achievable by suppressing unnecessary communication may have only limited impact in increasing the lifetime of the node. Alternatively, in order to limit the overall power consumption, the network could schedule for sensing only a subset of the nodes and activate other units only if the deviation among readings at different sensors exceeds some accuracy threshold (e.g., 5dBA, as discussed is section 5.3.2).

In the scenario described above, the nodes are regularly spaced and thus spatial sensor selection techniques like the ARS cannot be applied. However, in scenarios in which mobile phones are opportunistically used to collect sensor data in urban areas [38, 177], the ARS, or a modified version thereof, may be applicable. For instance, the average loudness of a busy street or a park could be estimated by sending a corresponding query to mobile phones located within the area of interest. If the density of devices is high, like it may be the case in a busy street, letting all the available phones answer the query may cause a high amount of traffic to be generated. Using the ARS it is possible to reduce the number of responses while still providing the application with a sufficient number of noise readings. If the mobile phones are aware of the average number of devices present within their "sensing area", they can decide upon their participation in the sensing through the ARS strategy, possibly using a ranking based on the local density only (see sections 4.5.1 and 4.7). [10]

5.4. Capturing Noise Levels Using Wireless Sensor Nodes

Performing accurate noise level measurements requires the availability of adequate hardware sensing platforms. To understand the feasibility of wireless sensor nodes to be used as noise pollution sensing devices, we thus tested and evaluated three different hardware platforms. At a preliminary stage, we considered using the Tmote Sky platform from Moteiv [136] equipped with the SBT80 multi-modality sensor board available from EasySen[11]. As reported in [167] and section 5.4.1 below, however, we rapidly abandoned this platform due to its high inaccuracy. We then evaluated the performance of the Tmote invent sensor

[10] A mobile phone can autonomously estimate the local density of devices in its neighborhood leveraging, for instance, its bluetooth radio [60].

[11] `www.easysen.com`

node, also from Moteiv [165]. Although this platform allowed us to gather more reliable measurements of noise levels, it still shows serious limitations, like the lack of adequate calibration procedures for the on-board microphone (cf. section 5.4.2). We thus further evaluated the possibility of using commercially available noise level meters, adequately interfaced with the Tmote Sky sensor node, to perform noise level measurements in WSNs. Based on our experimental results, we believe this third option to be the most suitable, although also the most expensive, to implement the application scenario under consideration (cf. 5.4.3). In the following, we provide detailed results of the experimental evaluation of the three mentioned platforms.

5.4.1. The SBT80 Sensor Board

The SBT80 sensor board features, among other sensors, the EM6050P-423 omni-directional condenser microphone, which we used to capture audio signals from the environment. The output voltage of the microphone is quantized using the 12 bits analog to digital converter (ADC) of the Tmote Sky. The voltage levels recorded by the microphone may be reconstructed from the ADC samples using the formula: $E = \frac{E_{ADC}}{4096} \cdot V_{ref}$, where V_{ref} is set to 2.5V for the Easysen sensor board and E_{ADC} are the ADC samples. Figure 5.1 reports the (A-weighted) voltage response of the microphone to a synthetically generated 250Hz sine wave stimulus, whose amplitude has been progressively increased to bring the microphone to saturation. The raw signal samples, read from the ADC at a a 2kHz rate, have been sent from the node to the a personal computer through the serial port. We processed the data in Matlab to compute the A-weighted instantaneous acoustic pressure $p_A(t)$, which is proportional to the voltage levels depicted in figure 5.1.

The first consideration we can derive by observing the sample data in figure 5.1 is the high level of background noise. The EM6050P-423 has indeed a self-noise level of 54 dB(A),[12] which makes sounds corresponding to noise levels below 54 dB(A) to be indistinguishable from electrical background noise.[13] The EM6050P-423 turned out to have further suboptimal characteristics. For instance, its frequency re-

[12] The self-noise (or equivalent noise) level may be computed by subtracting the nominal signal to noise ratio (SNR) from the reference sound pressure level (SPL) of 94dB. For the EM6050P-423 the SNR is 40 dB.

[13] 54 dB(A) is the noise level that can be measured during a conversation taking place at about 1 m distance from the microphone.

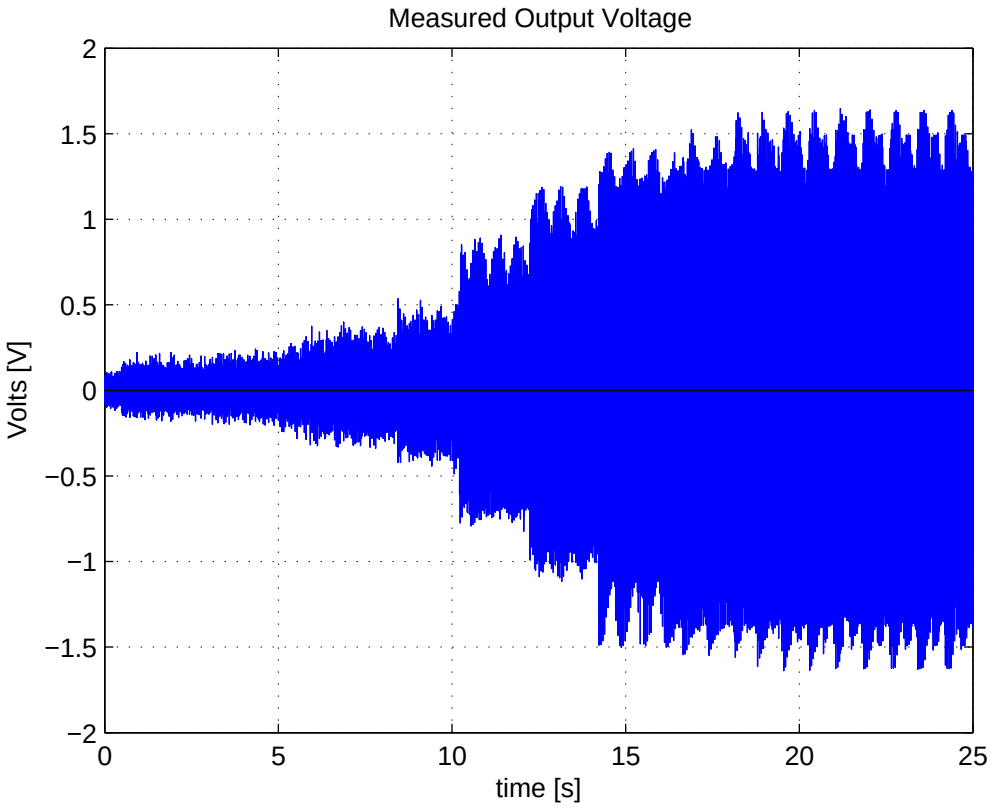


Figure 5.1.: Response of the microphone of the SBT80 sensor board to a 250Hz sine wave stimulus of increasing amplitude.

sponse starts deviating from linearity already at 5kHz and significantly distorts harmonic components above 10 kHz. The lack of proper signal conditioning (e.g., frequency filtering) on the SBT80 board makes this distortion enter (and thus tamper with) the computation of the noise levels. Due to these limitations, the SBT80 sensor board does not represent a suitable sensing device for the assessment of noise pollution levels.

5.4.2. The Tmote invent

The Tmote invent prototyping platform, available from Moteiv [136], provides an extended sensor suite including a high-quality, omnidirectional electret microphone. To capture noise levels using this platform, we implemented a dedicated TinyOS application that collects raw acoustic samples, computes the corresponding equivalent noise levels (using a remotely settable time period T), and reports the computed values to a central sink at regular time intervals. Our application runs on top of Boomerang [137], Moteiv's proprietary distribution of the

TinyOS operating system, and computes linear noise levels, thus without applying the A-weighting filtering.

To investigate the performance of the Tmote invent prototyping platform as a noise pollution sensor we run several measurement sessions in both indoor and outdoor settings. In our experiments, up to eight Tmote invent nodes concurrently collected raw acoustic samples (at a rate of 8 kHz) and computed the corresponding equivalent noise level with a temporal granularity T of one second. The nodes could communicate directly with the data sink and reported noise level readings every T seconds. The sink node, physically attached to a computing device (a laptop), immediately forwarded incoming messages to TinyLAB, our Matlab-based tool for interaction with WSNs (see section 6.1 for more details). A TinyLAB application then collected messages from the sink node, timestamped, processed and visualized the measured noise levels in real-time, and finally stored the data in Matlab format. Figure 5.2 shows an example of a measurement session along an urban street. In the following, we comment about the results of our experiments. In particular, we discuss the possibility of using the Tmote invent to detect and count vehicles on an urban road, as well as the crucial issue of microphones calibration. Further, we report on a design defect of the Tmote invent platform, which we happened to come across during our experiments.

Vehicles identification and counting. To gain first insights into the performances of the Tmote invent platform, we deployed the sensor nodes along an urban road (at about 3 meters distance) and recorded node responses. Figure 5.3 shows a segment of the collected data with the typical rises of the equivalent noise level values caused by vehicle transits. The rises are respectively labeled with the actual type of vehicle passing by, which we manually annotated during the experiment. This data shows, for instance, that the high noise rise produced by a bus transit extends over a longer period of time if compared to that produced by a car.[14] This characteristic, along with additional information like magnetometric data, could be exploited to design a detector able to count total vehicles transits and possibly differentiate between different

[14]Figure 5.3 also shows noise levels produced by trams to be below those corresponding to buses or cars. This counterintuitive discrepancy is due to the fact that the tram tracks ran along a different street, which was about two hundreds meters away from the measurement point. The cars and buses recorded were passing on the directly adjacent road, as close as three meters away from the nodes.

Figure 5.2.: Measurement session with the Tmote invent.

vehicles categories [7]. However, since we are mainly interested in the computation of noise levels, we did not further investigate this issue.

Calibration. To observe the behavior of different nodes in response to the same acoustic stimuli, we deployed 4 Tmote invent nodes in a silent indoor environment at close distances from each other. We then used the *Audacity*[15] tool to produce a chain of 5 seconds wide white noise pulses of increasing amplitudes. Figure 5.4 shows the responses of the 4 nodes to these acoustic events, clearly pointing out a misalignment in the measured equivalent noise levels. We believe this discrepancy to be mainly due to mismatches in the sensitivity of the microphones of the different nodes. Indeed, the sensitivity of a microphone may deviate from the nominal value due to flaws in the manufacturing process, experienced mechanical shocks, or temperature gradients [24]. At least the last two issues are likely to have affected the sensitivity of the microphone of the used sensor nodes. To cope with this problem, microphones should be regularly re-calibrated, so as to realign their

[15] `audacity.sourceforge.net`

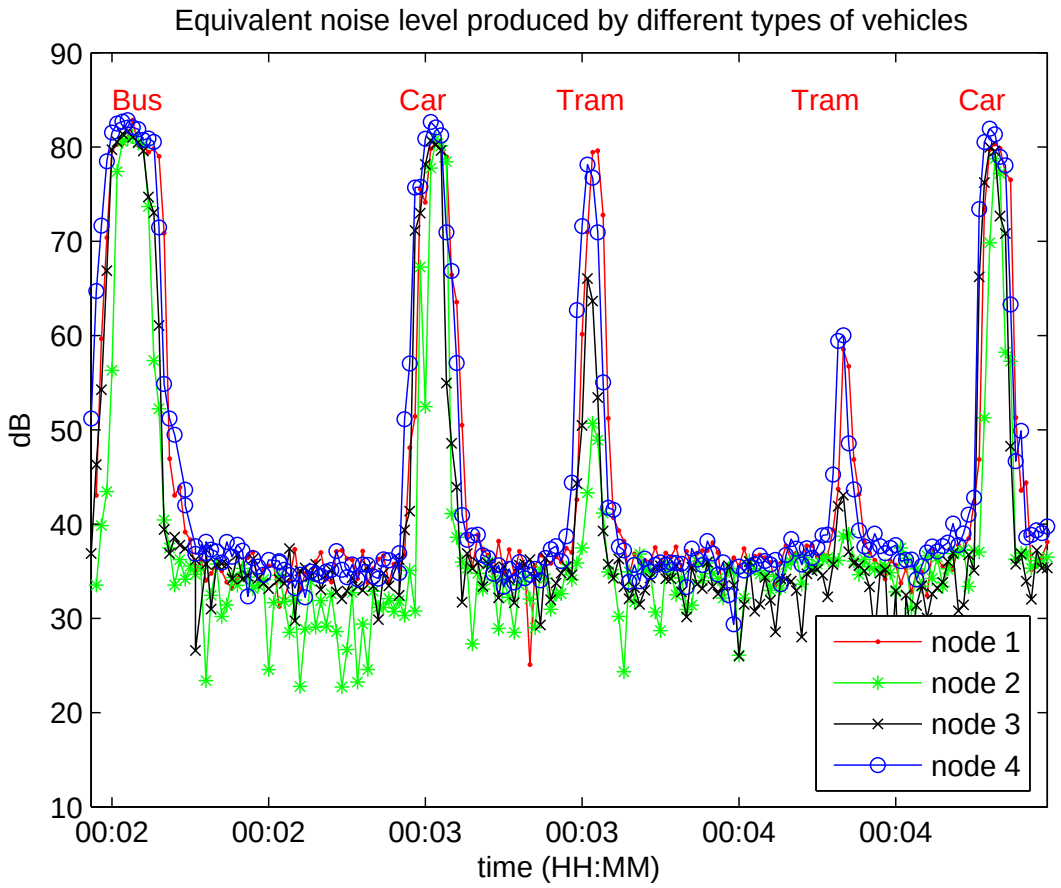


Figure 5.3.: Acoustic responses of four different nodes in correspondence of vehicles transits.

responses with respect to a reference level. However, developing an adequate calibration procedures for the Tmote invent would involve modifications of the platform at the hardware level. To avoid such cumbersome and prone-to-failure intervention, we decided to resort to commercially available noise level meters and properly interface them with the sensor node, as described in the next section 5.4.3. Notwithstanding, we believe our quantitative study of the performance of the Tmote invent to represent a useful contribution for other researchers interested in measuring noise levels using WSNs. Furthermore, as detailed below, our experiments allowed us to detect and document a case of design defect of the Tmote invent platform.

Platform defect. While analyzing noise data collected using the Tmote invent, we observed an unexpected behavior of the reference voltage of the nodes. In particular, we noticed that under a constant acoustic stimulus the average output voltage of the microphone assumes differ-

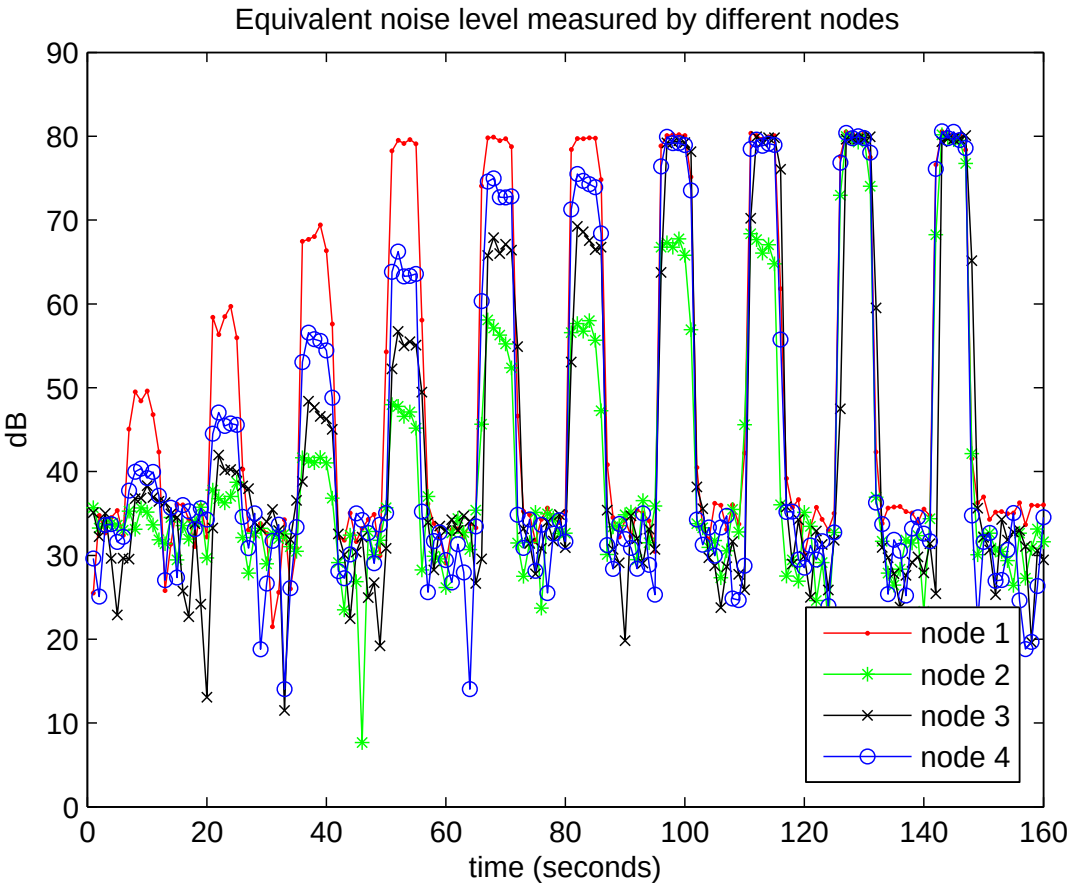


Figure 5.4.: Acoustic responses of four different nodes to a chain of white noise pulses of increasing amplitude.

ent values depending on the node being plugged into a power outlet or draining current from its own batteries. Since we observed a perfectly analogous behavior for all the eight Tmote invent platforms we used in our experiments, we report results related to a single, representative, sensor node. Figure 5.5 helps illustrate the above mentioned malfunctioning by reporting in subplot (a) the development of the total number of samples collected during a single sampling interval, in subplot (b) the average output voltage of the microphone and in subplot (c) the computed equivalent noise level. We annotated different sectors of the plot with letters from a to h, to identify different phases of our experiment. As observation begins, the node is attached to a power outlet through an appropriate USB adapter[16], and the average output voltage of the microphone is 0.8 volts (sector a). Once the node is

[16]Since the Tmote invent features a USB (series A) plug, it can be attached to a power line using an appropriate adapter. To ensure that the problem described here was not due to the adapter itself, we performed several experiment using different units of the same adapter, as well as using different adapter models.

detached from the adapter, this voltage level increases up to 1.1 volts, as shown in sector b of figure 5.5(b) and regularly returns to 0.8 volts if the node is plugged in again (sector c). Surprisingly, the effect of plugging/unplugging the node is also visible on the number of samples collected during the interval T, as shown in subplot 5.5(a). Indeed, as long as the node is plugged into the power outlet it collects about 8200 samples, while this figure increases up to 8600 samples once the node runs on batteries.[17] This oscillation does not (appear to) significantly influence the computed equivalent noise levels, but clearly indicates a malfunctioning in the circuitry regulating the power supply. Furthermore, we could observe a far more annoying malfunctioning if, instead of unplugging the node from the USB adapter, both the node and the adapter are detached from the power outlet. In this case, the average output level of the microphone decreases to a very small value, which hinders proper computation of noise levels, as shown in subplots 5.5(b) and 5.5(c) (sector d). Instead, the number of samples keeps oscillating as observed above (see subplot 5.5(a)). Plugging in the node does not help in repairing the malfunctioning microphone (sector e) and only a node reboot restores the initial node behavior (sector f). Sectors g and h of figure 5.5 finally show the reproducibility of the above described behavior in the case the sensor node is plugged in the USB adapter (sector g) or not (sector h). Although this behavior is not expected to influence nodes' operation in the field, where they usually cannot be plugged in and out from a power outlet, it definitively represents a problem to be aware of when performing lab-scale experiments.

We would also like to point out that the use of the TinyLAB tool allowed us to identify and easily analyze this unexpected behavior of the reference voltage of the nodes. Before deploying the network in an outdoor environment, we indeed tested the hardware and software in our lab to both understand signal dynamics and investigate the issue of calibration. The comfortable and powerful visualization and processing features offered by the Matlab computing environment, made available by the TinyLAB tool, enabled a fast and comfortable real-time analysis of the data reported by the sensor nodes. Analyzing the behavior of the Tmote invent platform using a typical approach in which data is collected, stored and analyzed at a later stage, would have most likely delayed our prototyping process considerably.

[17] Since the interval T extends for one binary second (1024 binary milliseconds), a sampling rate of 8 kHz results in 8192 samples/second.

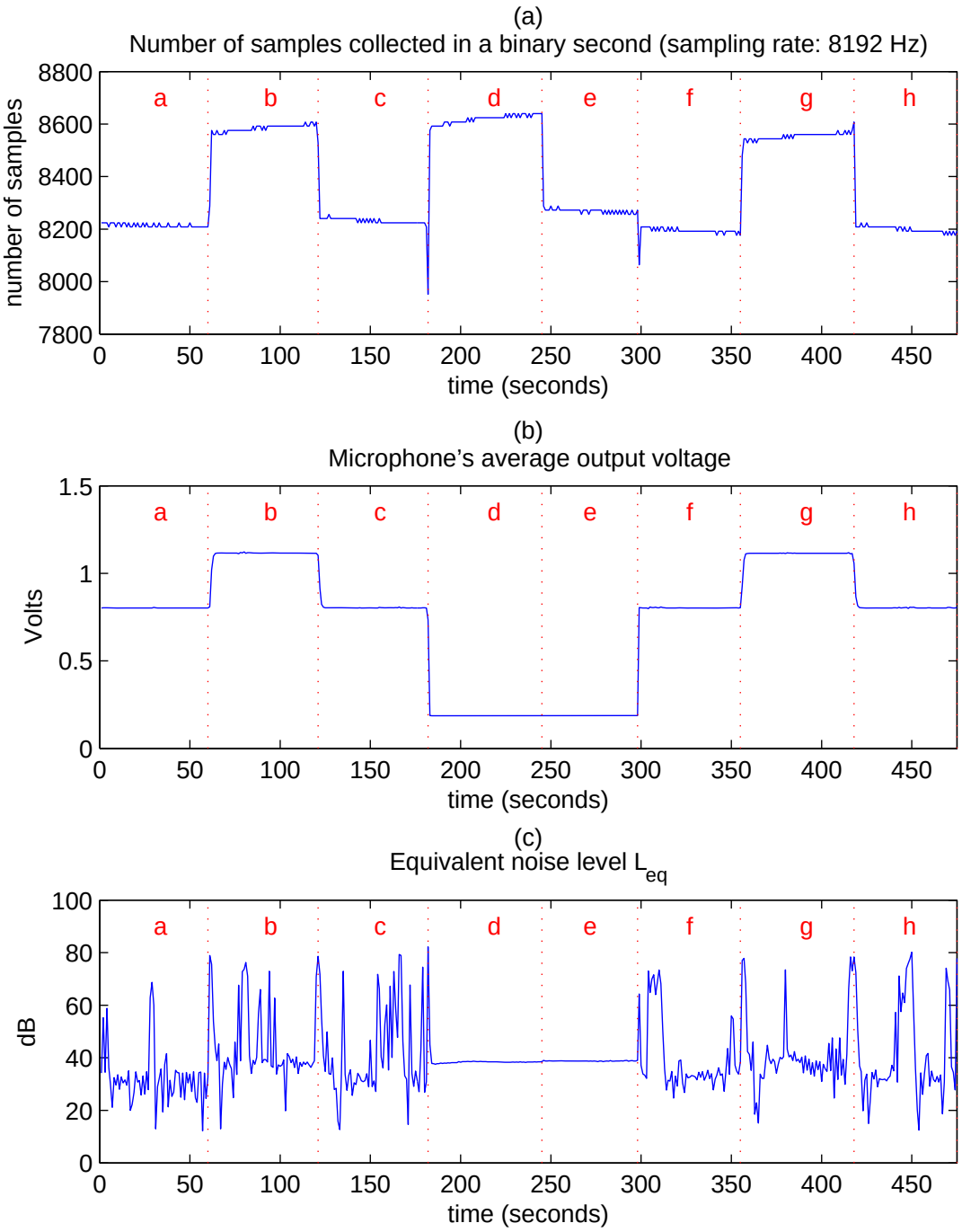


Figure 5.5.: Tmote invent's behavior with different power supplies.

5.4.3. Tmote Sky and Noise Level Meter

The above reported observations regarding the lack of calibration of the Tmote invent platform show that using cheap, generic sensor node platforms to capture noise levels may result in the collection of unreliable, and thus unusable, data. Furthermore, properly sampling the acoustic pressure levels measured by the microphone requires the sensor nodes to continuously sustain high sampling rates (8 kHz in our experiments). The data acquisition process can therefore drain a significant amount of processing power and hamper the concurrent execution of other tasks, like radio communication.

To cope with the above mentioned issues while still exploiting the potential of WSN systems, we investigated the possibility of interfacing the Tmote Sky platform with off-the-shelf sound level meters. There exists a large number of such devices, with different characteristics with respect to form factor, costs, and accuracy. We thus surveyed available products looking for a device able to guarantee good accuracy but limited cost and form factor. To this end, we mainly focused on noise level meters able to comply with the class-2 international standard,[18] which are adequate for the general field use we are targeting, and are significantly cheaper than class-1 devices. Further, we looked for devices providing adequate output channels so as to enable a direct connection to the Tmote Sky and avoid the need of developing customized hardware interfaces.

We found the Extech 407740 class-2 sound level meter to be a good candidate for our purposes.[19] This device makes the measured noise levels available on an analog output channel. The signal on this channel can be easily gathered using a standard 3.5 mm TRS-connector (audio jack), connected to pins 9 and 10 of the 10-pin expansion connector of the Tmote Sky.[20] With this connection, gathering noise level data from the Extech 407740 simply requires polling the proper ADC channel of the Tmote Sky at the desired sampling rate. Note that the Extech 407740 is not an *integrating-averaging* but an *exponential time-weighted* sound level meter. As discussed in section 5.3.1, devices belonging to the first category measure the equivalent noise level $L_{Aeq,T}$ over a

[18] The requirements noise level meters must fulfill to be certified as class-1 or class-2 devices are included in the IEC 61672-1 standard [91].

[19] `www.extech.com/instrument/products/400_450/407740.html`

[20] Pin 10 is the analog ground, while pin 9 corresponds to one of the four ADC (Analog-to-Digital Converter) channels available on the 10-pin expansion connector of the Tmote Sky.

period T, which represent the integral over T of the square of the instantaneous acoustic pressure $p(t)$, properly scaled and A-weighted. Instead, an exponential time-weighted sound level meter measures the instantaneous values of the A-weighted and time-weighted sound level, $L_{A\tau}(t)$, defined as [91, page 13]:

$$L_{A\tau}(t) = 10 \log_{10} \frac{1}{\tau} \int_{-\infty}^{t} \frac{p_A^2(\xi)}{p_0^2} e^{-\frac{(t-\xi)}{\tau}} d\xi. \tag{5.2}$$

Thus, the output of the device is an exponentially weighted average of the squared instantaneous acoustic pressure. The value of the time-weighting constant τ can be either $125ms$ (F-weighting) or $1s$ (S-weighting), depending on the measurement situation [91]. In most settings, however, the value of τ should be set to $1s$ [55]. By sampling the $L_{A\tau}(t)$ at regular intervals of extension $\leq \tau$ and then properly integrating these values over the interval T, it is possible to compute an approximation of the $L_{Aeq,T}$. In order to enable a meaningful comparison of data collected by different devices, certified calibrators like, e.g., the Extech 407744, should be used.

In the next section 5.5, we demonstrate the use of the *Tmote Sky + Extech 407740* platform. In particular, we apply it as a reference noise level meter for evaluating the performance of mobile phones as noise pollution sensors.

While the Extech 407740 provides high fidelity in it measurements, it has an unfavorable form factor. Although it is a hand-held device, its dimensions (25.5x7x2.8 cm) are considerably bigger than those of the Tmote Sky (6.5x3.2x1.3cm). Looking for alternative solutions, we found that the main board of the Dostmann SL328 sound level meter[21] only measures 4.2x2.5x0.5cm, and could thus be easily mounted on top of the Tmote Sky. However, the SL328 does not provide standard output channels to capture the measured noise levels. Therefore, its use requires the development of a customized hardware interface to the Tmote Sky. Although we have investigated the design of such interface, we do not have a working prototype at our disposal.

5.5. Capturing Noise Levels Using Mobile Phones

The use of mobile phones as generic, ubiquitous sensing platforms gathered increasing interest in the last years [38, 60, 102, 130, 133, 177]. How-

[21] www.dostmann-electronic.de/PHP/docs/233.pdf

ever, using mobile devices for the assessment of environmental noise levels poses several technical challenges.

For instance, information about the context of the user should be collected and possibly used to trigger or inhibit data collection [197]. It would indeed be of little value to perform a measurement while, e.g., the phone is in a pocket or in a bag.

Further, access to the resources of the mobile phone should occur in the background, possibly without requiring the user to perform any action to participate in the sensing task [112]. In particular, access to hardware and software resources should not hamper the concurrent use of the phone for "traditional" applications, like short text messaging messaging applications. Additionally, adequate primitives to allow the user to set her privacy settings should be available. Indeed, since measuring noise levels requires performing audio measurements, users may mistrust the application and fear for their conversations to be recorded in the background. Besides supporting users' privacy from the technical side (e.g., by computing noise levels on the phone and thus transmitting only averaged data to the back-end server), providing adequate incentives for participation in the sensing task may help reducing or redirecting users concerns.

Last but not least, the actual accuracy of the collected noise measurements should be known. Although it is to be expected that off-the-shelf mobile phones will not be able to reach the accuracy of dedicated sound level meters, it is important to ascertain if the obtainable accuracy is sufficient to allow the envisioned applications to work reliably. In many cases, for instance for the identification of quiet bike trails [38], it could be sufficient to infer discrete states from the raw measurements (e.g., *quiet*, *moderately loud*, or *very loud*). However, the specific hardware and software characteristics of mobile phones may prevent this possibility. For instance, audio input channels of mobile phones typically feature noise canceling, low-pass filters, and/or dynamic input level adjustment. The presence of these processing stages may hamper the possibility of measuring the actual absolute loudness of an acoustic signal. Furthermore, since microphones of mobile phones are obviously not intended for noise measurements, calibration issues arise, as in the case of wireless sensor nodes.

To investigate the feasibility of mobile phones to be used as noise pollution sensors, we performed a series of experiments for which we describe the setup and the corresponding results below.

5.5.1. Experimental Setup

Our experiments aim at investigating two main issues, namely, the comparableness of the acoustic measurements of nearby located phones and the accuracy of such measurements against those taken by a reference sound level meter. To this end, we used three Nokia N95 8GB mobile phones[22] as test devices. In the following, we refer to these devices as *Phone1*, *Phone2* and *Phone3*. The Nokia N95 8GB well represents a state-of-the-art mobile phone, featuring significant processing power, good permanent storage capability, and several built-in sensors, like an accelerometer and a GPS-receiver. We emphasize here that the availability of a GPS-module is an essential feature for devices supporting mobile sensing applications, since it allows to associate a sensor value with the corresponding location at which the measurement took place. Furthermore, recent work demonstrated that accurate indoor-localization is possible by opportunely processing signals received by standard Bluetooth and WLAN modules [25], both being available on the Nokia N95 8GB devices.

We used three devices of the same type so as to to avoid discrepancies in the responses of the mobile phones being due to differences in the built-in hardware. Although we experimented on devices of a specific manufacturer (Nokia) and type (N95), our conclusions qualitatively apply also to most commercially available mobile phone platforms, since these usually exhibit comparable characteristics with respect to their audio circuitry and software.

In our experiments, we stimulated the mobile devices with a series of three different acoustic signals (the *tones*, the *chirp*, and the *traffic* test signal) and recorded the responses of the phones for off-line analysis. We provide a description of these test signals along with the discussion of our experimental results below. To record the responses of the mobile phones, we implemented two simple applications, one in Python (PyS60) and the other in Java (J2ME), which capture audio signals and store them in a *wave* file. Both the Python and the Java API expose simple methods for recording audio data, but while Python only allows to capture signals at a rate of 8kHz, the Java API enables sampling rates of up to 41kHz. Lower level primitives granting direct access to the unprocessed raw audio data are in both cases still unavailable. This implies that before we can record them, the audio

[22] `www.forum.nokia.com/devices/N95_8GB`

signals likely undergo the processing steps typical for voice communication applications, like band-pass filtering, noise canceling, or input level adjustment. This "polishing" of the signal, if not bypassed or properly accounted for, constitutes a serious burden to the use of mobile phones for the assessment of environmental noise levels, as we also discuss in section 5.5.2 below. The third programming option for the Nokia mobile devices under consideration is C++ Symbian. The corresponding API offers methods for setting low-level audio parameters like the microphone gain, as well as for defining custom codecs or selecting the specific audio source (built-in microphone, line-in, phone call or radio). To the best of our knowledge, however, C++ Symbian does not expose methods for immediate access to the raw audio data, as also pointed out in [130].

In all our experiments, the Extech 407740 class 2 professional sound level meter (hereinafter also called the *phonometer*), connected as described in section 5.4.3, has been co-located with the three mobile devices and collected ground-truth noise level measurements. We set the phonometer to gather data using F-weighting in time and A-weighting in frequency. F-weighting basically runs a moving average on the squared acoustic pressure using a time constant of 125 ms, as discussed in section 5.4.3. We correspondingly applied F- and A-weighting filters also to the responses of the mobile phones and indicated the resulting A-weighted noise levels in dB(A) (A-weighted decibels). We then averaged the collected readings over an interval T to compute (an approximation of) the equivalent noise level $L_{Aeq,T}$.

To reproduce the test signals we used a common laptop supporting up to 192kHz audio in-/output that we connected to high quality external speakers. In the following, we will refer to this system as the "audio source".

5.5.2. Experimental Results

Response to the *tones* test signal. Our first experiment aimed at studying the response of the mobile phones to synthetic test signals produced by an audio source in a controlled environment. We aligned the three mobile devices and the phonometer on a surface in a silent room[23], all at approximately the same distance from the audio source, as shown

[23] When the audio source is off, the average noise level in this room, measured with our reference phonometer, is slightly above 30dB(A).

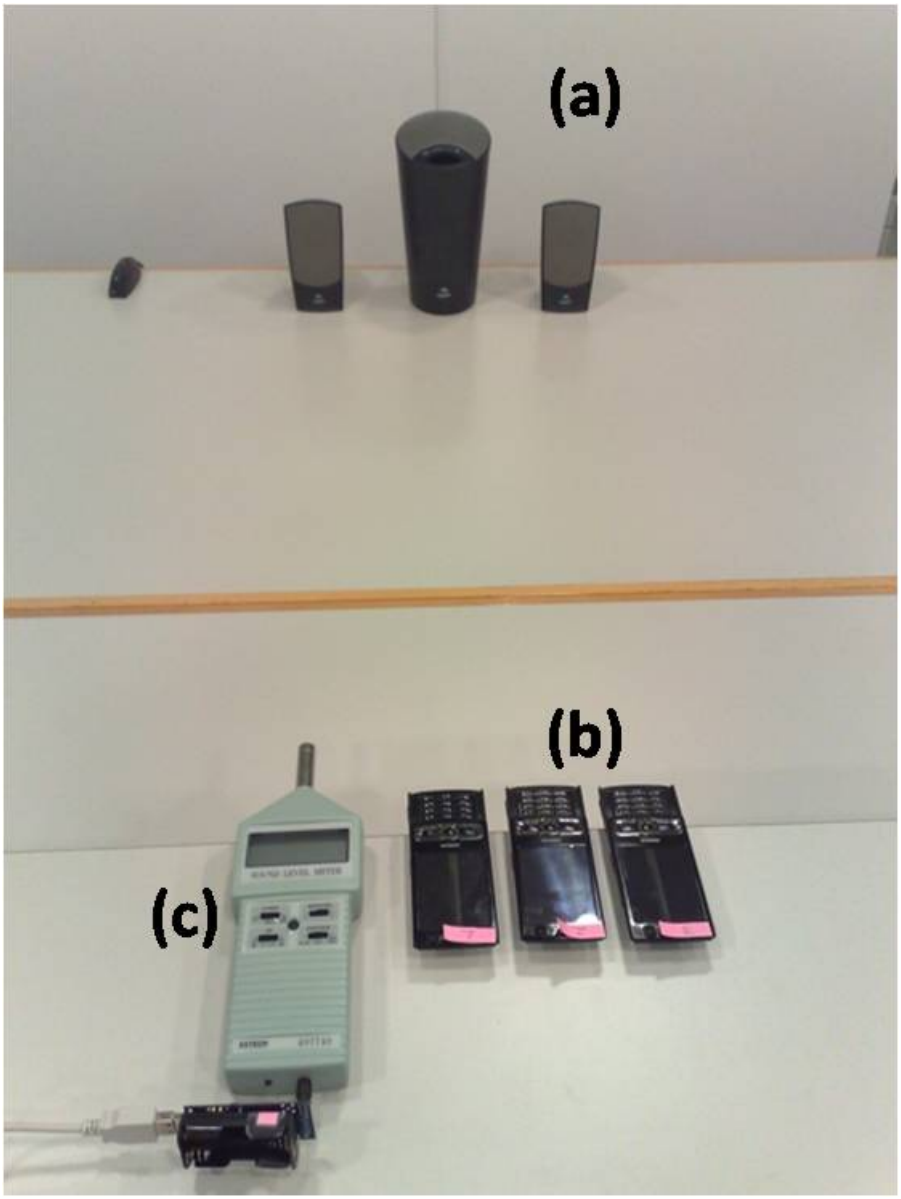


Figure 5.6.: Experimental setup: **(a)** the audio source; **(b)** the three mobile phones; **(c)** the phonometer.

in figure 5.6. In this setting, we reproduced three times a one minute long test signal and recorded responses of the phonometer and the three mobile phones. The test signal starts with a five seconds long white noise snippet, followed by five seconds of *silence* (i.e., the audio source is on but outputs a zero-amplitude signal). After this first phase, the test signal repeats five times a five seconds long 1 kHz sinusoidal tone, whose amplitude is regularly increased at each repetition (varying from 20% to 100% of the total available output dynamic). The tones are interleaved with five seconds long silence cuts. We use a pure tone as the test signal since standard calibrators calibrate sound level meters

Figure 5.7.: Response of *Phone2* to the first run of the *tones* test signal.

at one single frequency, namely 1 kHz. Since the signal is a repetition of sine tones, we will refer to it as the *tones* test signal. Figure 5.7 displays an example of the recorded instantaneous acoustic pressure levels corresponding to a single run of the *tones* test signal. As we can see, the mobile phone properly mirrors the increasing amplitude of the five sinusoidal tones. Instead, the response of the mobile phone to the white noise signal, framed in a black rectangle in figure 5.7, clearly shows the effect of a noise canceling filter. After recording the white noise signal for about one-third of the five seconds snippet, the filter classifies the signal as background noise and suppresses it, causing a reduction in the amplitude of the recorded signal, and, consequently, a diminution of the "perceived" loudness associated with the signal. Noise canceling is a standard signal processing technique used in several audio applications and it is therefore not a surprise to see its effect in this measurement. However, these results clearly demonstrate that built-in noise canceling algorithms can hamper the ability of mobile phones to measure environmental noise levels, since they may partially suppress the very signal one is actually trying to capture. As mentioned above, the availability of adequate programming primitives granting access to

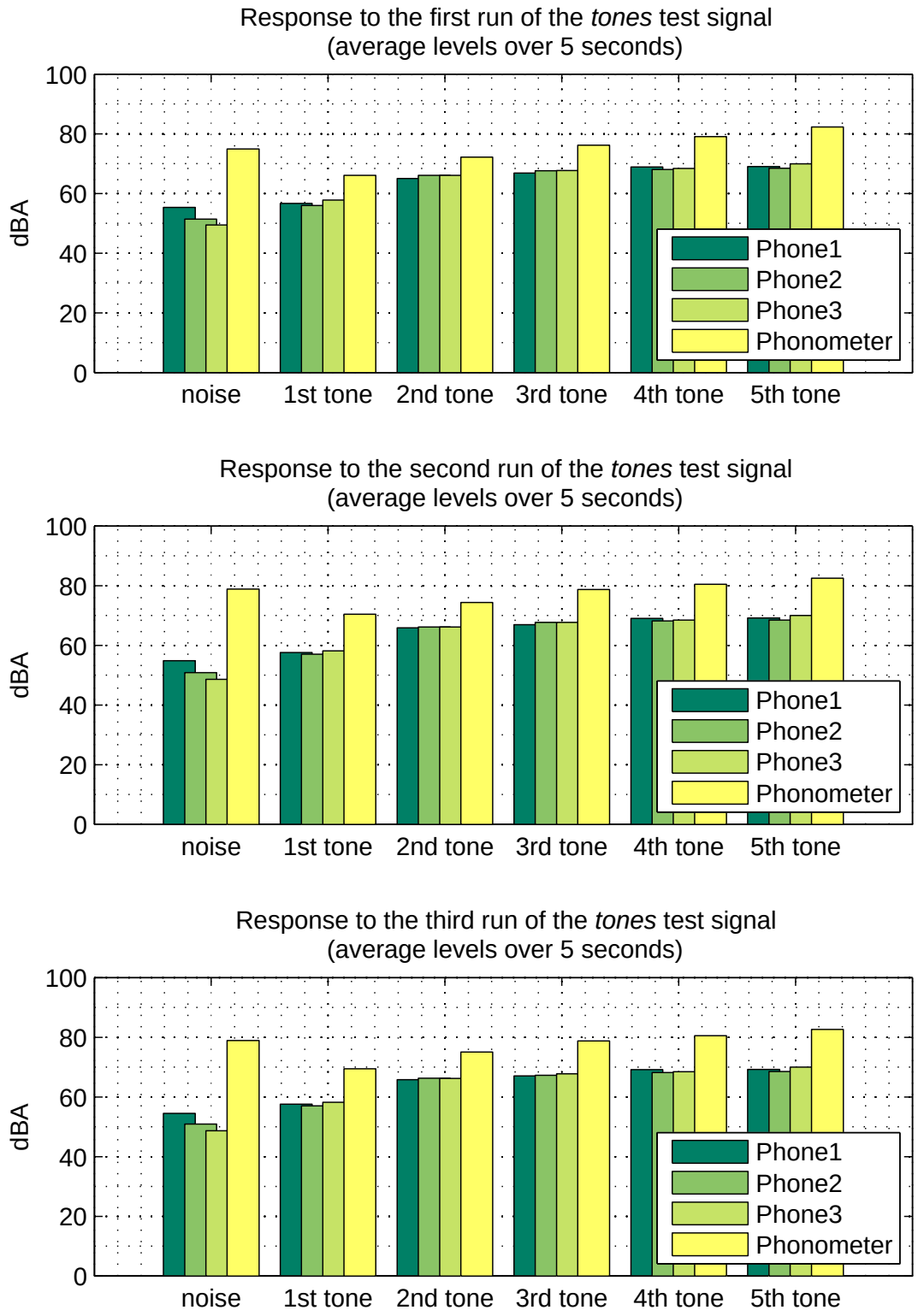


Figure 5.8.: Response to the three runs of the *tones* test signal, for the three devices under test *Phone1*, *Phone2* and *Phone3* and the reference phonometer.

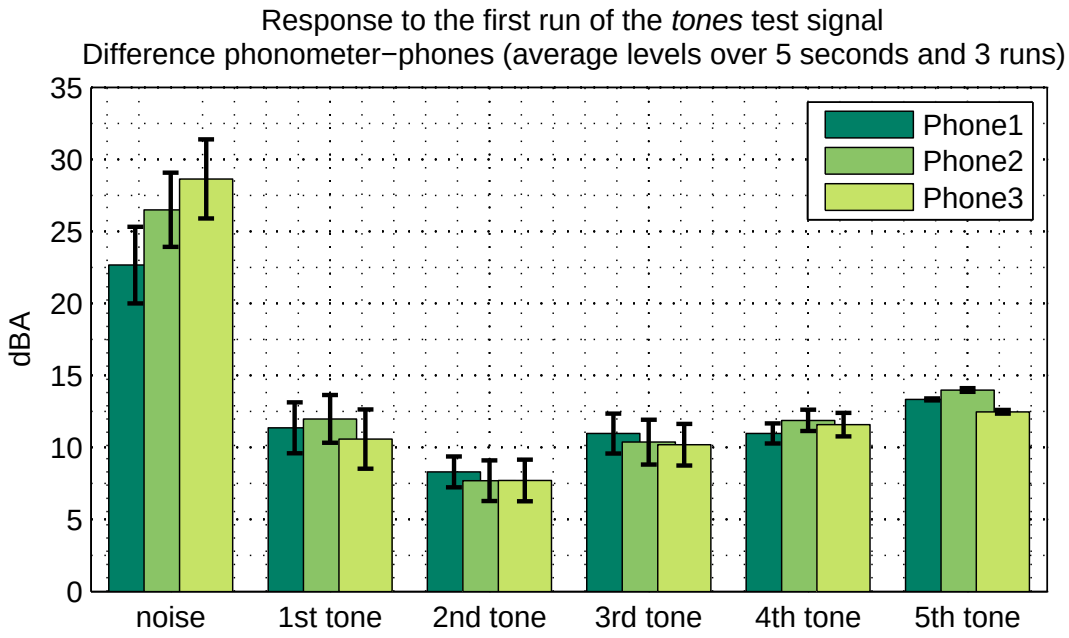


Figure 5.9.: Response to the first run of the *tones* test signal. Comparison of the difference of the levels measured by the phonometer and the corresponding levels captured by *Phone1*, *Phone2* and *Phone3*. The difference are averaged over the three runs of the test signal.

the raw audio data would allow to bypass this problem, but this is still not possible with the currently available APIs.

The noise levels corresponding to the recorded acoustic pressure levels, properly A- and F-weighted and averaged over five seconds, are reported in figure 5.8, along with the noise levels measured by the phonometer. Figure 5.9 reports the differences between the measurements of the phonometer and those of the three phones for the first run of the tones *test* signal. Both figures 5.7 and 5.9 make evident that, when the test signal is white noise, there is a high discrepancy (> 20dB(A)) between the noise level measured by the phonometer and those measured by the phones. This discrepancy is consistently lower, although still high (~ 10dB), for all five subsequent measurements and for all the three test devices. A constant divergence from the reference measurement would represent a correction factor that could be used for calibrating the mobile phones against the reference itself. However, our experiments showed that this value may change by several (> 5) dB(A)s when the measurement is repeated in the same setting but at a different time.

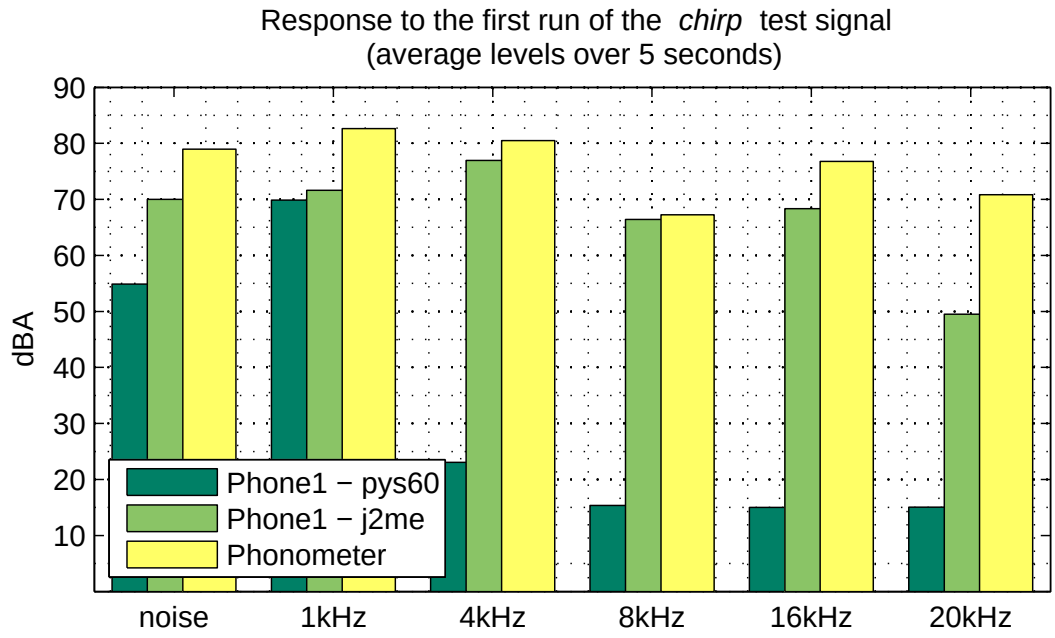
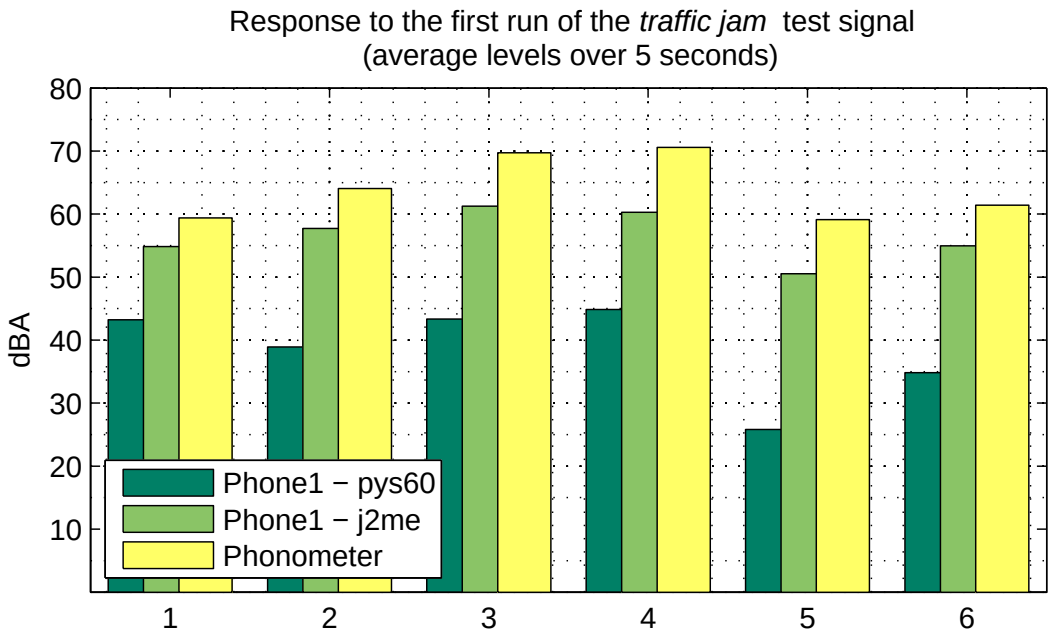
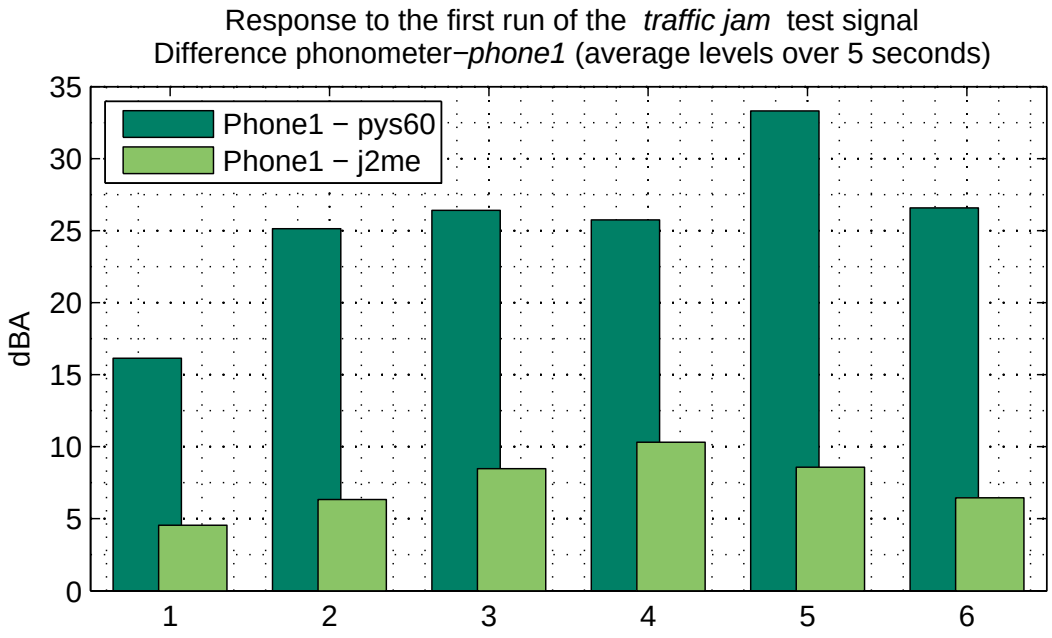


Figure 5.10.: Equivalent noise levels measured by the phonometer and by *Phone1* (using both the PyS60 and J2ME applications). The levels are averaged over the 5 seconds duration of each tone of the *chirp* test signal.

Response to the *chirp* test signal. To investigate the ability of the mobile phones to capture signals at different frequencies, we performed a second experiment using a test signal consisting of five sine waves of equal amplitude whose frequencies increments from 1 to 4, 8, 16 and 20kHz. Since a signal whose frequency content increases linearly with time is typically called a *chirp*, we refer to this sequence of sine tones as the *chirp* test signal. Using the same experimental setting as before, we played the sixty seconds long test signal three times in a row and recorded the responses for offline analysis. Figure 5.10 shows the average noise levels measured during the first run of the *chirp* test signal, using both the PyS60 and the J2ME applications and along with the corresponding values recorded by the phonometer. These levels result from averaging the F- and A-weighted squared instantaneous acoustic pressure over the five seconds long snippets of the *chirp* test signal. As we can see, the audio signal captured by the PyS60 application is clearly low-passed, since already the response to the 4kHz tone is so feeble that it approaches the value measured when no test signal is present. This results doesn't come as a surprise, since for voice transmission applications a low-pass filtering at a frequency around 4 kHz is a standard figure. However, since the human hear can actually hear frequencies far above 4 kHz [24], bypassing this low-pass filter is manda-



(a)



(b)

Figure 5.11.: Equivalent noise levels measured by the phonometer and by *Phone1* using both the PyS60 and J2ME applications (a), and relative differences (b). The levels are averaged over 5 seconds snippets of the *traffic* test signal.

tory if the mobile device is intended to be used as noise level meter. The J2ME programming primitives clearly allow to access richer audio input data, since our J2ME application manages to capture, although with low accuracy, also the tone at 20 kHz, as shown again in figure 5.10. This superior recording quality, however, comes at the cost of higher processing and storage loads.

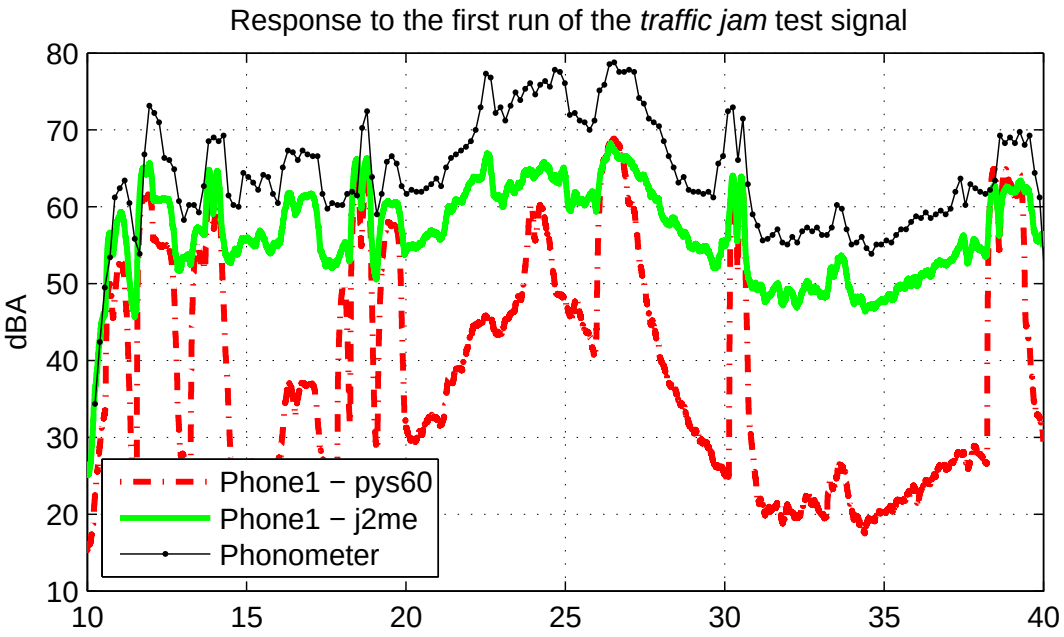


Figure 5.12.: Response of the phonometer and *Phone1* to the first run of the *traffic* test signal.

Response to the *traffic* test signal. To evaluate the response of the mobile devices in more realistic conditions, we performed a third experiment using a thirty seconds long recording of busy street traffic. The signal features several different honks, a few screeching breaks, and plenty of engine noises, resulting in complex frequency and amplitude spectra. The experimental setting is the same as in the first two experiments and the test signal is again played three times in a row. Since the responses to the three subsequent runs of the test signal are very similar to each other, we discuss only the results ascertained in the first run. Since the PyS60 and the J2ME applications cannot run concurrently on the mobile phone, we performed the test twice, recording once with the PyS60 and once with the J2ME application. As expected, since the test signal and the experimental setting did not change at all, the

phonometer profiles collected during the two subsequent experiments overlap almost perfectly. For this reason, in commenting figures 5.12 and 5.11 we can indistinguishably refer to *the* phonometer profile. A visual inspection of figure 5.12 shows the relatively high fidelity of the J2ME-profile when compared to the actual ground-truth measurements collected by the phonometer. The PyS60-profile, on the other hand, diverges consistently from the phonometer profile, as also the average noise levels reported in figure 5.11(a) show. The difference between the average levels recorded by the phonometer and those captured by *Phone1* are also reported in figure 5.11(b). This graph illustrates the higher accuracy reachable using the J2ME application, but also shows that the difference between the noise levels measured by the phones and those measured by the phonometer vary significantly across different time sectors of the signal.

The experimental results presented in this section allow us to make some general qualitative assertions about the feasibility of using mobile phones as noise pollution sensors. First, devices of the same type return coherent audio responses so that a direct comparison of their readings is, at least qualitatively, possible. However, the measured noise levels may diverge from those captured by a co-located sound level meter and the magnitude of this divergence may vary significantly depending on the actual audio signals. Moreover, the accuracy of the measured noise levels is influenced by the processing steps the signal undergoes before being recorded. Further, the specific language used to program the devices may influence the accuracy of the measured noise levels, since different APIs may expose audio data at different "rawness" levels. Therefore, it is difficult to provide a generic characterization of the actual accuracy with which mobile phones can capture noise level measurements. This makes these devices bad candidates to provide for noise mapping data, which must be reliable. However, mobile phones may still be useful to support applications with less stringent accuracy requirements, provided their limitations are adequately taken into account.

5.6. Summary

In this chapter, we introduced and analyzed an example application scenario for WSNs: the monitoring of environmental noise in urban areas. We motivated our choice on this specific scenario by showing its poten-

tial relevance for the enforcement of European environmental protection policies. Further, we provided an in-depth analysis of the application requirements and discussed the applicability of both temporal and spatial sensor selection. Since we identified the choice of suitable sensing platforms to be one of the main challenges towards the implementation of this application, we evaluated several candidate platforms. In particular, we first focused on wireless sensor nodes and provided experimental results showing the issues that hamper the use of generic WSN prototyping platforms in this application context. We then suggested using commercially available noise level meters, opportunely interfaced to the Tmote Sky sensor node, as viable sensing platforms. Further, we investigated the reliability of noise level measurements collected using mobile phones. In this context, we showed that, at the current state-of-the-art, these platforms cannot provide noise data with the accuracy required for the purpose of noise mapping.

6. Tools and Libraries

Before presenting our conclusions and discussing possible directions for further research in the next chapter 7, we provide here a description of the tools and libraries we developed in the context of this work. In particular, we first focus on the TinyLAB tool [162], which we used extensively to support the design and testing of the temporal sensor selection algorithms reported in chapter 3. The TinyLAB tool was also instrumental in facilitating the evaluation of wireless sensor network (WSN) platforms for noise monitoring, as discussed in chapter 5. Second, we describe our TinyOS implementation of the ES-AMS, which has been introduced in sections 3.4, 3.5, and 3.6. All the software artifacts developed in the context of this thesis are publicly available from the author's website[1].

6.1. TinyLAB

The first steps towards a wireless sensor network deployment often include a preliminary stage in which sensor data is collected, visualized and carefully analyzed. In this stage, developers often undergo a time-consuming procedure logging data first and analyzing it later with standard or ad-hoc tools. The possibility to visualize and process the sensor data and interact with the network (e.g., to change the current sampling rate) in real-time can ease and speed-up the preliminary data analysis, as well as support debugging and network inspection at a later stage. To this end, several tools have already been designed and developed within the WSN research community [30, 72, 78, 202]. However, many of these tools are not publicly available. Furthermore, such tools are usually tailored to specific applications or needs and thus provide, by design or for lack of implementation, limited functionalities. On the other hand, existing general purpose scientific computing platforms could be leveraged to support data visualization and processing in the context of WSN. For instance, the Matlab computing

[1] As of October 2009 hosted at `http://people.inf.ethz.ch/santinis/`.

environment, introduced in section 2.2.3, is widely used across different scientific communities and, being a generic data managing platform, appears suitable to be used also in WSNs settings. Indeed, the TinyOS software suite includes a collection of Matlab scripts that allow to access and use the TinyOS Java toolchain. These scripts provide basic primitives to interact with a sensor network from within Matlab. However, this solution requires binding Matlab to the TinyOS tools and thus limits flexibility and portability.

To overcome these limitations, we developed TinyLAB, a software framework completely implemented in Matlab that allows to receive and send messages from and to a TinyOS-based sensor network. Avoiding any cumbersome installation procedure, TinyLAB enables using the full Matlab computing power to manage incoming messages, process, store and visualize data as it comes from the network. Furthermore, TinyLAB allows to send control messages to specific nodes, groups, or the whole network. The TinyLAB software is publicly available along with application examples and a users guide. In the following, we show the main functionalities of TinyLAB on the basis of a simple test application.

6.1.1. Writing TinyLAB Applications

To handle incoming and outgoing packets, TinyLAB relies on two basic abstractions, which we named *PacketSource* and *PacketSink*. A *PacketSource* basically wraps a communication channel, like a serial or TCP/IP port, and provides *PacketSink*s with properly parsed packets. A *PacketSink* declares interest in all or specific messages coming from a *PacketSource* and defines the payload parsing modalities as well as further operations to execute on the incoming data.

For instance, figure 6.1 shows the code of the *testTinyLAB* application, which receives packets from the serial port $COM5$, as specified by the *sourceDescriptor* at line 11. The *PacketSource* is instantiated at line 14, and will start receiving and forwarding packets after the correspondent *startReceiving* function has been called (line 27). The received packets are forwarded to the *PacketSink* instantiated at line 17, which is bound to the previously defined *PacketSource* at line 20. However, a *PacketSource* forwards packets only according to the specific interests declared by its *PacketSink*s. For instance, figure 6.1 (line 17) shows that 4 options have been specified to generate the *PacketSink*

```
1  %TESTTINYLAB runs the testTinyLAB application
2  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3  % testTinyLAB
4  % author:    silvia santini (santinis@inf.ethz.ch)
5  % created:   yesteryear
6  % last modified:
7  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
8
9  % SET UP ENVIRONMENT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
10
11 sourceDescriptor = 'serial@COM5:tmote'; %'sf@localhost:9001';
12
13 %create a new packet source
14 [psource,psourceIndex] = PacketSource('new',sourceDescriptor);
15
16 %create a new packet sink
17 psink = PacketSink([76,77],'this','complete','testTinyLABReceive');
18
19 %bind the sink to the source
20 PacketSource('bind',psource,psink);
21
22 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
23
24 % START LISTENING TO INCOMING PACKETS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
25
26 %Incoming packets will be processed in the function testTinyLABReceive
27 startReceiving(psource);
28
29 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
30
31 % SEND MESSAGES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
32
33 msg = newMessage('testTinyLABReceiveMsg');
34 msg.sourceID = hex2dec('FEFA');
35 msg.code = 2; %code = 1 -> toggle blue led
36               %code = 2 -> toggle red led
37               %code = 3 -> toggle both blue and red leds
38
39 send(psource,'TOS_BCAST_ADDR',msg);
40
41 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figure 6.1.: The *testTinyLAB* application.

of the *testTinyLAB* application. The first declares that the *PacketSink* is only interested in receiving messages whose *AMtype*[2] is either 76 or 77. The second and third options allow to specify whether the application is interested in receiving the raw packets or their appropriately parsed version. Finally, the last option allows to specify that received packets must be forwarded to the *testTinyLABReceive* function, which will then process them according to the application requirements. Finally, lines 33-39 show the use of TinyLAB's *send* primitive, which allow to inject packets into the network.

[2] The AM type is an identifier that specifies the active message type of the payload of TinyOS packets [89].

In order to receive and parse packets, TinyLAB must know at least their raw structure. In particular, the *AMtype* and the payload length must always be specified. If no additional information is given, however, TinyLAB will only be able to handle the received messages as a sequence of bytes. Nonetheless, if the number and type of fields included in the payload are specified, TinyLAB can parse the received bytes according to this structure. To this end, the message format must be specified in a dedicated file and included in the same folder of the application code. This, in turn, basically requires to import and adapt the message definition files of the TinyOS packets the application is interested in. For instance, figure 6.2 shows an excerpt of the TinyOS file *testTinyLABMsg.h* that defines the fields of the data payload of the messages received by TinyLAB's *testTinyLAB* application. TinyLAB's counterpart of the *testTinyLABMsg* is shown in figure 6.3.

```
1  struct testTinyLABMsg {
2    uint16_t sourceMoteID;
3    uint16_t epochCounter;
4    uint16_t currentData;
5  };
6
7  struct testTinyLABreceiveMsg {
8      uint16_t sourceMoteID;
9      uint8_t code;
10 };
11
12 enum {
13   AM_TESTTINYLABMSG = 76,
14   AM_TESTTINYLABRECEIVEMSG = 77
15 };
```

Figure 6.2.: Definition of the payload of the TinyOS messages received by TinyLAB's *testTinyLAB* application.

Further details on how to implement TinyLAB applications are included in the documentation that is delivered along with the code, as mentioned above.

6.2. A TinyOS Library for Adaptive Model Selection

In chapter 3 we introduced our AMS algorithm and provided its experimental evaluation for two different sets of models, namely autoregressive and exponential smoothing models. In particular, we implemented both the AR-AMS and ES-AMS in Matlab and evaluated their per-

```
1  struct testTinyLABMsg {
2    uint16_t sourceMoteID;
3    uint16_t epochCounter;
4    uint16_t currentData;
5  };
6
7  struct testTinyLABreceiveMsg {
8      uint16_t sourceMoteID;
9      uint8_t code;
10 };
11
12 enum {
13   AM_TESTTINYLABMSG = 76,
14   AM_TESTTINYLABRECEIVEMSG = 77
15 };
```

Figure 6.3.: TinyLAB's definition of the payload structure of a *testTinyLABMsg* packet.

formance on 20 data sets retrieved from real WSN deployments. Our evaluation showed the ability of the AMS to provide for a good model choice for implementing the DPS strategy. Furthermore, we pointed out that the very limited computational and memory overhead required to run and update ES models makes the ES-AMS suitable to be executed on resource constrained sensor nodes. To corroborate this claim, we implemented the ES-AMS in TinyOS and tested its performance on a lab-scale deployment, as we discussed in section 3.6. In the following, we provide a more detailed description of our implementation. We point out at this stage that we first wrote an implementation of the ES-AMS, named *esAMS*, for TinyOS 1.x. We then ported it to TinyOS 2.x in the context of a student project [108]. However, since the experimental evaluation reported in section 3.6 has been performed using the first version of the *esAMS*, we focus here of this TinyOS 1.x implementation.

The *esAMS* is a TinyOS application that periodically collects sensor readings and reports them to the sink using a DPS strategy. The model used to run the DPS is chosen among a set of double exponential smoothing models (DES), as discussed in section 3.5.1. To this end, we include in the set of candidates several DES models corresponding to different values of the smoothing constants α and β. In particular, we let α and β vary, with step 0.1, within the intervals $[0.1, 1]$ and $[0, 1]$, respectively. Thus, the number of models in the set of candidates is $N_{AMS} = N_\alpha \cdot N_\beta = 10 \cdot 11 = 110$, where N_α and N_β represent the considered number of different values of the parameters α and β, respectively.

As mentioned in section 2.2.2, a TinyOS application is composed of a set of components that are wired together through standardized interfaces. The *esAMS* embeds several standard components offering functionalities to sample one or more sensors at regular time intervals as well as sending and receiving messages. In particular, we leverage the TinyOS interfaces *Timer*, *ADC*, *SendMsg*, and *ReceiveMsg*. The *Timer* interface provides basic primitives to generate one-shot and periodic timers, which are used to control the duration of the initialization phase and of the sampling interval. The *ADC* interface allows to retrieve readings from an opportunely wired sensor. Finally, the *sendMsg* and *receiveMsg* interfaces can be leveraged to transmit and receive messages to and from the sink, respectively. The code of the *esAMS* application is encapsulated in the three nesC files named *esAMSC.nc*, *esAMSM.nc*, and *esAMSMsg.nc*. They provide for the wiring configuration, the actual application code, and the message formats, respectively.

```
1  includes esAMSMsg;
2
3  configuration esAMSC { }
4  implementation
5  {
6    components Main, esAMSM
7               , TimerC
8               , LedsC
9               , HumidityC as Sensor
10              , GenericComm as Comm
11              , DelugeC
12              ;
13
14   Main.StdControl -> DelugeC;
15   Main.StdControl -> esAMSM;
16   Main.StdControl -> TimerC;
17
18   esAMSM.InitializeTimer -> TimerC.Timer[unique("Timer")];
19   esAMSM.SamplingTimer -> TimerC.Timer[unique("Timer")];
20   esAMSM.Leds -> LedsC;
21   esAMSM.SensorControl -> Sensor;
22   esAMSM.ADC -> Sensor.Temperature;
23   esAMSM.CommControl -> Comm;
24   esAMSM.ReceiveMsg -> Comm.ReceiveMsg[AM_esAMSMRECEIVEMSG];
25   esAMSM.DataMsg -> Comm.SendMsg[AM_esAMSMSG];
26 }
```

Figure 6.2 shows the esAMSC.nc configuration file, which wires the interfaces used by the esAMS application to their actual TinyOS implementations. In particular, we see that the *esAMS* uses two different instances of the *Timer* interface, named *InitializeTimer* and *SamplingTimer* (lines 18 and 19 in figure 6.2). These timers are used to set the length of the initialization phase and the length of the sampling interval,

respectively. Further, line 22 in figure 6.2 shows that the *ADC* interface is wired to the *Temperature* interface exposed by the *HumidityC* component. This connection allows to access the external temperature sensor of the Tmote Sky platform (see also section 2.2.1). Using any other available sensor only requires to appropriately adapt this part of the configuration file and does not require any change in the code implementing the actual application logic.

The esAMS application supports two kinds of messages, the *esAMSMsg* and *esAMSreceiveMSg*, whose definition is included in the file *esAMSMsg.nc*. Model updates are sent within a *esAMSMsg* packet, which must in principle only include the values of the parameters L and b for the current model. Indeed, equation 3.18 shows that using DES models the sink only needs to know L and b in order to predict future sensor readings. Besides the current model the *esAMSMsg* packet may also include additional information like, e.g., the current sample, or the values of the smoothing constant α and β corresponding to the current model. An *esAMSreceiveMsg* packet includes a single field representing the code of a specific operation the user wants the node to execute. For instance, in our implementation the reception of a *esAMSreceiveMsg* packet with operation code 1 makes the node revert into the initialization phase. *esAMSreceiveMsg* messages can be sent to one, several, or all the nodes in a network. To this end, we used the TinyLAB tool described in the previous section.

Figure 6.2 shows the structure of the *esAMSM.nc* file, which includes the actual application logic. The code from line 11 to 20 declares the interfaces used by *esAMSM.nc*, which are linked to actual implementations through the configuration specified in esAMSC.nc. After this declaration, the actual implementation of the component starts with the definition of the used variables and data structures, as commented at line 24. Then, the commands *init()* and *start()* of the *StdControl* interface are called. The command *init()* initializes the communication, sensor, and timer components, and allows to set variables and status flags to their default values. Then, control goes to the *start()* command, which activates the communication and sensor components and starts the *InitializeTimer*. When this timer fires, the event handler *InitializeTimer.fired()* (line 38) is executed. The main role of this handler is to trigger the first data collection by calling the command *ADC.getData()*, which, in turn, interrogates the sensor actually wired to the ADC interface (the external temperature sensor, in our case). As soon as the

ADC holds a stable sample, it generates the event *ADC.dataReady* (line 42), which makes the current sensor reading available to the program. The *ADC.dataReady* event handler first stores the current sample in a global variable and then checks, by looking at the value of an appropriate globally visible *INITIALIZE* flag, if the AMS must undergo an initialization phase. In this case, it posts the *initializeAMS()* task (line 46), otherwise it posts the *AMS()* task (line 48). We also use a dedicated flag to signal whether all the operations related to a single run of the AMS are completed or not. This flag, which we named *AMSdone*, is set to false in the *ADC.dataReady* handler, before any AMS-related task is called. The *initializeAMS()* task simply initializes the parameters of all models in the set of candidates. For DES models, this corresponds to setting the value of L_1 equal to the current sample and $b_1 = 0$, irrespective of the values of α and β (see also equation 3.19). The virtual models are initializes in the same way and the relative update rate is set to 1 for all models. Further, the *INITIALIZE* flag is set to false, so that the next occurrence of the *ADC.dataReady* event will cause the *AMS()* task to be called. Finally, the *initializeAMS()* task posts the *sendUpdate()* task, which selects the current model h^* and sends it to the sink. Further, the *SamplingTimer* is started. This periodic timer will fire at regular time intervals until it is stopped by a call to the command *SamplingTimer.stop()*. This happens if either the AMS must be re-initialized or if an unrecoverable problem occurs (like, e.g., if the *AMS()* task does not manage to run to completion before the next sample is ready for processing). Thus, each time the *SamplingTimer* fires, a new data is collected from the ADC and processed in the *ADC.dataReady* handler. As we described above, unless initialization is required, this handler only stores the current data and posts the task *AMS()*. This, in turn, increases the samples counter and computes the estimation of the current sample using the model h^* that had been previously communicated to the sink. It then posts the task *updateModels()* which takes care of updating the parameters of all models in the current set along with their virtual counterparts, as explained in section 3.4.3. The task *updateModels()* also computes the relative update rate for all models at each sampling round. After posting the task *updateModels()*, the *AMS()* task checks if the current prediction error exceeds the tolerated threshold. If not, it just sets the *AMSdone* flag to true and completes its execution. Otherwise, if first posts task

selectModel() and, immediately after, the *sendUpdate()* task.[3] The task *selectModel()* (line 52 in figure 6.2), individuates, among the set of candidates, the model with the lowest relative update rate and selects it as the new current model. The task *sendUpdate()* then sets the fields of the *esAMSMsg* structure described above with the appropriate values. Since this operations signals the end of the current run of the AMS, the *AMSDone* flag is accordingly set to true. Then *sendUpdate()* calls the *send* command of the interface *DataMsg* that hands over the current packet to the underlying communication layer. If this operation succeeds, and thus the packet is successfully scheduled for transmission, the *DataMsg.sendDone* event returns successfully ((line 56 in figure 6.2)). Line 60 in figure 6.2 also shows the presence of an event handler for processing received packets. Within this handler, we implemented the logic necessary to make some of the variables of the applications remotely controllable. For instance, we can dynamically change the value of the error threshold or require the AMS to re-initialize. Furthermore, we also included in our implementation the possibility to run in a special *debug mode*, in which an *esAMSMsg* message is sent at each sampling round, irrespectively of the current prediction error. This is achieved using a *DEBUG_FLAG* that, when set to true, triggers the sending of an update but does not influence the further operation of the AMS. Using a specific control message, we can also makes motes dynamically switch from debug to normal operation mode (and vice versa) at any point in time.

[3] We should recall here that TinyOS1.x tasks are run to completion before the next task is executed. This ensures the *sendUpdate()* task to be executed only after the *selectModel()* runs to completion.

```
includes esAMSMsg;
#include <stdlib.h>

/**
 * Module esAMSM
 *
 * Implements the DPS using the ES-AMS strategy for model selection
 */
module esAMSM {
  provides interface StdControl;
  uses {
    interface Timer as SamplingTimer;
    interface Timer as InitializeTimer;
    interface Leds;
    interface SplitControl as SensorControl;
    interface ADC;
    interface StdControl as CommControl;
    interface SendMsg as DataMsg;
    interface ReceiveMsg;
  }
}
implementation {

  /**declare here data structures and tasks */

  command result_t StdControl.init() { return SUCCESS; }

  event result_t SensorControl.initDone() { return SUCCESS; }

  command result_t StdControl.start() { return SUCCESS; }

  event result_t SensorControl.startDone() { return SUCCESS; }

  command result_t StdControl.stop() { return SUCCESS; }

  event result_t SensorControl.stopDone() { return SUCCESS; }

  event result_t InitializeTimer.fired() { return SUCCESS; }

  event result_t SamplingTimer.fired() { return SUCCESS; }

  async event result_t ADC.dataReady(uint16_t data) {
    return SUCCESS;
  }

  task void initializeAMS() { }

  task void AMS() { }

  task void updateModels() { }

  task void selectModel() { }

  task void sendUpdate() { }

  event result_t DataMsg.sendDone(TOS_MsgPtr sent, result_t success) {
    return SUCCESS;
  }

  event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr m) { return m; }
}
```

7. Conclusions and Outlook

In this final chapter, we summarize our contributions, outline the main limitations of our approaches, and sketch interesting directions for future work.

7.1. Contributions

In the context of this thesis, we investigated the sensor selection problem in distributed, wireless sensing systems. In particular, we focused on typical environmental monitoring scenarios requiring long-term, periodic sensor data collection. Drawing and improving upon existing work, we designed practical temporal and spatial sensor selection strategies, which allow to reduce the overall energy consumption of the network while guaranteeing the quality of the collected data. To this end, we avoided or limited the use of a priori information about the signals of interest. Furthermore, we designed our sensor selection strategies to be suitable to run on resource-poor wireless sensor network (WSN) platforms. To assess the performance of our algorithms, we conducted both simulation studies and experiments on a small-scale WSN deployment.

In the context of temporal sensor selection, we investigated the use of the dual prediction scheme (DPS) as a simple but powerful means to provide for communication savings while guaranteeing the collected data to lie within a pre-specified error threshold. We initially proposed an implementation of the DPS based on the least-mean-square adaptive filter, which was able to achieve more than 90% of communication savings, with respect to the default monitoring scheme, on selected data sets. We then discussed the main drawback of this approach, common to other implementations of the DPS, consisting in the fact that all or part of the parameters of the used prediction model are set a priori. This may result in a lack of flexibility of the model and can, thus, hamper the achievable communication savings. To cope with this problem, we thus designed the AMS, a generic framework for online parameter estimation and model selection. We provided two different implemen-

tations of the AMS, one based on autoregressive models (AR), named AR-AMS, and one leveraging exponential smoothing models (ES), accordingly referred to as the ES-AMS. We evaluated both implementations on several data sets retrieved from real WSN deployments and demonstrated the ability of the AMS to provide for high communication savings without requiring any a priori information about the signals of interest. Further, we showed that the ES-AMS achieves comparable performance, in terms of communication savings, with respect to the AR-AMS, but incurs in lower computational and memory overhead. This makes the ES-AMS particularly suited for an implementation on WSN platforms. We thus also provided the ES-AMS as a TinyOS application and tested its behavior on a small-scale WSN deployment.

As for spatial sensor selection, we focused on sensor field reconstruction applications and worked under the assumption that the ACT algorithm is used at the central server to process the collected readings. In this context, we showed that the problem of selecting proper sampling geometries may be reduced to a coverage problem. We thus focused on existing coverage preserving protocols and seeked after improving their performance in terms of communication overhead. In particular, we introduced distance-based sensor ranking heuristics that allowed to significantly reduce the communication overhead of the CCP coverage configuration protocol. Further, we leveraged the same heuristics to design an adaptive random sensor selection strategy (ARS) that can provide for high levels of coverage of the region of interest while activating a significantly lower number of nodes with respect to a simple random selection strategy. We then briefly addressed the issue of cross-layer optimization and provided a preliminary study to investigate the possible interplay of our ARS strategy with the CTP data collection protocol.

Finally, we discussed the applicability of our sensor selection strategy to a specific application scenario, and provided a description of the tools and libraries implemented in the context of this work.

7.2. Limitations and Future Work

The sensor selection approaches presented in the previous chapters and summarized above, present some limitations as well as potential for further improvements, as discussed below.

7.2.1. Adaptive Model Selection

The AMS algorithm allows to select a suitable model to implement the DPS data collection strategy. To this end, a set of candidate models is maintained on sensor nodes and their performance are continuously evaluated over time. In section 3.4.2 we further introduce the racing method, which allows to prune bad performing models from the set of candidates. Eventually, the racing mechanism may let only a single model in the set of candidates, but this would clearly hamper any further adaptation to the actual data dynamics. Therefore, we suggested to abort the racing procedure once the cardinality of the set of remaining models reaches a desired size. Although we did not further investigate this issue, adequately refining the racing procedure may allow to reduce the computational overhead and memory footprint of the AMS while still preserving its adaptive nature.

One of the main limitations of the AMS consists in its potentially slow response to changes in the data dynamics, when used over longer periods of time. For instance, assume a model h_1 performs significantly better than all other models in the set of candidates during the first phase of data collection. After a given period of time, the performance of model h_1 may start deteriorating, while other models, e.g., h_2, could provide for higher communication savings. However, due to the "advantage" in terms of relative update rate accumulated by model h_1 in the first period, it may take a long time for model h_2 to finally become the current model. This is a clear inefficiency of the AMS, which could however be corrected in at least two ways. First, one could introduce a threshold for the maximum allowed gap between the performance of the current model and those of other candidate models. This would ensure a maximum "delay" in the identification of a new best current model. Second, it could be possible to observe the differential performance of the current model with respect to its candidate peers. If the performance of the current model shows a negative trend for more than a given number of time steps, an adequate "realignment" procedure could be triggered. For instance, one could simply re-initialize the AMS and thus allow potentially new best performing models to come into play. The introduction of such procedures, however, could make the AMS become unstable, and should thus carefully designed.

A general limitation of the DPS data collection procedure consists in the fact that it requires a reliable communication link between the

node and sink to be available. Thus, in case of messages losses, the sink considers as reliable possibly wrong estimations of the current sensor readings. Since such losses may and do occur in WSNs, removing this assumption would allow for a more flexible applicability of the DPS. To this end, we briefly mention an idea that may provide a useful hint for further research. In particular, assume an estimation of the average packet loss ratio p_{loss} of the link between source and sink to be available. Then, an equivalent error threshold $e^{eq}_{max} = f(p_{loss}, e_{max}) \leq e_{max}$ could be defined as a function of p_{loss} and e_{max}, and used as the error threshold in place of e_{max}. If the e^{eq}_{max} is properly defined, its use may allow to guarantee the sensor readings estimation error to lie, on average, within the threshold e_{max} even in presence of message losses. We believe that properly using the prediction intervals [122] of the chosen model may enable the computation of a proper value of e^{eq}_{max}.

7.2.2. Sensor Ranking

The experimental results reported in chapter 4 showed that our distance-based sensor ranking heuristics allow to improve the performance of the CCP coverage protocol. However, we performed our evaluation assuming the sensor nodes to be distributed uniformly at random over the region of interest. Although this is a frequent assumption in the WSN literature, we believe that considering other kind of network topologies may allow to provide for a more comprehensive evaluation of our approach.

Further, while studying the spatial sensor selection problem we assumed the value of the necessary spatial resolution Δ_s required to comply with the application requirements to be known a priori. In particular, we showed that using the ACT the value of Δ_s is linked to the value of the spatial bandwidth of the sensor field of interested. Since this value may change over time and vary over different sectors of the network, so should ideally do the value of Δ_s. Indeed, allowing for a local adaptation of the spatial resolution may allow to improve the quality of the final sensor field reconstruction. In particular, it would allow to distribute the sampling density according to the actual signal dynamics. Since the ACT features an automatic procedure to estimate the order M of the reconstructing polynomial (which, in turn, is related to the signal bandwidth and, thus, Δ_s), we believe such local adaptation procedure to be directly applicable in the sensor field re-

construction contexts we considered. For instance, the current local values of Δ_s could be estimated at regular time intervals by the central server, which would then distributed them to network.

Finally, we believe that the investigation of the cross-layer issues briefly discussed in section 4.8, constitutes a promising and interesting possibility for further research.

7.3. Concluding Remarks

The contributions reported in this thesis show that performing sensor selection in WSNs may allow for high communication savings even if no or little a priori knowledge on the signals of interest is available. To this end, the selection algorithms must be endowed with adequate automatic procedures providing for a timely adaptation to changing signal and network dynamics.

Thanks to their generic applicability and suitability to be implemented on resource-poor sensor nodes, we believe that our algorithms constitute practical solutions towards enabling long-term, efficient environmental monitoring using wireless sensor networks.

Appendices

A. The ACT Algorithm

In our approach to the field reconstruction problem in wireless sensor networks (WSNs), we refer to the ACT (Adaptive weights Conjugate Gradient Toepliz) algorithm as a reference technique for performing reconstruction from scattered samples [56, 75, 169, 180, 181]. Therefore, we provide here a summary of the main characteristics of the algorithm and discuss relevant related issues.

The problem of reconstruction from irregular samples can be defined as the problem of estimating a band-limited, or *essentially* band-limited, signal f from a set of irregularly spaced samples $f(s_j)$. The index j runs through some index set S_r of cardinality $|S|$ equal to r. The locations at which the samples are gathered may correspond to coordinates in a 1- or 2-dimensional space. In these cases, we indicate the s_j also as x_j, (x_j, y_j), respectively.

The problem of recovering a signal f from its samples $f_j, j = 1, \ldots, r$ is an intrinsically ill-posed problem, since the space of functions whose samples at the locations s_j corresponds to the values f_j has always infinite dimension. However, a-priori assumptions on the signal, like its band-limitedness, and the adoption of adequate numerical algorithms still allow for error-bounded reconstructions. The ACT-algorithm, for instance, allows to efficiently recover a signal from its irregular samples, even in presence of large gaps and clusters in the sampling set, as it may often be the case in WSNs scenarios. Other reconstruction procedures may work properly only for sampling sets resulting from a perturbation of a regular grid, require a maximal distance among samples to be guaranteed or be far less robust and efficient than the ACT [181]. In this context, we should also mention that the ACT can be applied to scenarios with irregular sampling geometries as well as those where the sampling set consists in a uniform grid with missing samples. Furthermore, an ad-hoc regularization technique allows to estimate the bandwidth of the signal to reconstruct on the fly, as reconstruction is performed [169]. The existence of such a procedure is particularly valuable, since it allows to correct possibly wrong bandwidth estimations

as well as to adapt to changing signal dynamics over time and space.

As we detail below, the computational efficiency as well as the reconstruction performance of the ACT is mainly determined by the characteristics of the sampling geometry. For instance, the number of collected samples r must be higher than a given threshold and the distance between a sample and its nearest neighbor must also oblige with a maximal allowed value. In a WSN scenario, it turns out that fulfilling these conditions requires the region of interest to be 1-covered by the nodes actively sampling the sensor field f. In this context, the sensing range R_s of the sensor nodes corresponds to the desired spatial resolution δ. In the following, we address this issue in more quantitative terms and motivate our choice of the ACT as a reference algorithm. For the sake of simplicity, we first introduce the basic equations of the ACT for both the 1- and 2-dimensional case in the following sections A.1 and A.2, respectively.

A.1. 1-Dimensional Case

In the 1-dimensional case, we assume the r locations at which the signal f has been sampled to be r points within the segment $[0, 1]$. Without loss of generality, we further assume that $0 \leq x_1 \leq x_2 \leq \ldots \leq x_r < 1$. The extension to the case in which the points are deployed over an arbitrary segment requires an appropriate normalization, as shown in [181].

The ACT algorithm basically solves the optimization problem of finding the trigonometric polynomial p^* of period 1 and order M that best approximates the function f. The value of M depends on the bandwidth of the signal and, for a discrete sequence of length N is always smaller than $N/2$. A trigonometric polynomial p is a finite linear combination of the sine and cosine functions. Using Euler's formula to express the trigonometric functions, the space of all trigonometric polynomials of degree M and period 1 is defined as:

$$P_M = \{p | p(x) = \sum_{k=-M}^{M} a_k e^{2\pi i k x}\}, \tag{A.1}$$

where the $a_k, k = -M, \ldots, M$ are the (complex) coefficients of the polynomial. As detailed in [75], there are several reasons while the choice of trigonometric polynomials is appropriate and convenient to

approximate a non-uniformly sampled band-limited function. Above all, however, there is the fact that it allows the reconstruction problem to gain structure and, thus, efficiency and stability [56, 75].

The problem of reconstructing the signal f from its samples thus turns into that of solving the least square problem:

$$\sum_{j=1}^{r} w_j |p(x_j) - f(x_j)|^2 = minimum\ in\ P_M, \tag{A.2}$$

where the weights $w_j, j = 1, \ldots, r$ allow to compensate for local variations of the density of the samples. To this scope, a possible choice for the w_js (also recommended in [56]) is:

$$w_j = \frac{x_{j+1} - x_{j-1}}{2}, j = 1, \ldots, r. \tag{A.3}$$

To compute w_1 and w_r one sets $x_0 = x_r - 1$ and $x_{r+1} = x_1 + 1$ [75]. The use of the weights w_j can significantly improve the stability and speed of converge of the algorithm and is therefore always recommended, in particular in presence of highly nonuniform sampling sets.

To find the optimal polynomial p_M^* that solves the problem A.2 the ACT defines the $(2M + 1)$x$(2M + 1)$ matrix T and the vector b of length $2M + 1$ with entries:

$$T_{lk} = \sum_{j=1}^{r} w_j e^{-2\pi i(l-k)x_j}, |l|, |k| \leq M \tag{A.4}$$

$$b_k = \sum_{j=1}^{r} w_j \widetilde{f}(x_j) e^{-2\pi i k x_j}, |k| \leq M, \tag{A.5}$$

Then, the set of $2M+1$ coefficients a_M^* that generates the polynomial p_M^*, is the solution to the linear system:

$$a_M^* = T^{-1} b, \tag{A.6}$$

The inversion of the matrix T can be accomplished through both direct or iterative methods, although the latter are usually preferred, especially in the presence of noise in the data. Since the matrix T has a Toeplitz structure, its inversion can be performed efficiently with either techniques. To speed up execution, the ACT inverts T via the conjugate gradient [70] iterative method. The name ACT actually summarizes the main features that give this reconstruction algorithm its efficiency and

robustness. First, the use of the adaptive (A) weights w_j that appear in equations A.2, A.4, and A.5. Second, the use of the conjugate (C) gradient method to solve the linear system A.6. Third, the Toeplitz (T) structure of the matrix T.

We should note that the matrix T depends, through the weights w_j and positions s_j, on the sampling geometry only. This means that we can control the properties of T by selecting appropriate sampling sets of nodes. Thus, tailored sensor selection strategies can be used to determine adequate sets of sampling nodes. As we will better detail below, both the existence and quality of a reconstructing polynomial p_M^* depend on the properties of T and, more specifically, on the value of its condition number[1].

A.2. 2-Dimensional Case

In the 2-dimensional case, the vector space of all trigonometric polynomials of order M (in both the x- and y-direction) is defined as:

$$P_M = \{p | p(x,y) = \sum_{m=-M}^{M} \sum_{n=-M}^{M} a_m n e^{2\pi i(mx+ny)}\}. \tag{A.7}$$

Accordingly, the least square problem A.2 assumes the form:

$$\sum_{j=1}^{r} w_j |p(x_j, y_j) - f(x_j, y_j)|^2 = minimum\ in\ P_M \tag{A.8}$$

where $(x_j, y_j), j = 1, \ldots, r$ are the x- and y- coordinates of the r sampling points in the unit square $[0,1]x[0,1]$ at which the samples f_j of the signal f are available. The natural choice for the weight w_j of sample s_j is the area of the Voronoi cell of s_j [75]. However, in most cases it is not necessary to estimate the exact value of the area and a gross estimate based on the local density would suffice, as shown in [75].

The matrix T and vector b can be computed extending the definitions A.4, and A.5. To take into account the increased dimensionality of the problem, the matrix T becomes a $(2M+1)^2$x$(2M+1)^2$ square matrix while the vector b has $(2M+1)^2$ entries. Thus, in the 2-dimensional case, we have:

[1] The condition number of a matrix A is defined as the ratio between the norm of the matrix and the norm of its inverse, i.e., $c(A) = \frac{\|A\|}{\|A^{-1}\|}$. For the computation of the condition number, usually the L_2 norm is used.

$$T_{kl,mn} = \sum_{j=1}^{r} w_j e^{-2\pi i((k-m)x_j+(l-n)y_j)}, |k|, |l|, |m|, |n| \leq M \qquad \text{(A.9)}$$

$$b_{kl} = \sum_{j=1}^{r} w_j f(x_j, y_j) e^{-2\pi i(kx_j+ly_j)}, |k| \leq M, \qquad \text{(A.10)}$$

The matrix T can also be rearranged in $(2M+1)x(2M+1)$ blocks B_{km} each of size $(2M+1)x(2M+1)$, thus $T_{kl,mn} = (B_{km})_{ln}$. Each of the B_{km} blocks is a Toeplitz matrix, and, thus, T is a block Toeplitz matrix [75].

Provided the matrix T is invertible, the solution to the least square problem A.8 can still be computed by the linear system 4.2. In the 2-dimensional case, however, the vector a_M^* has $(2M+1)^2$ entries.

A.3. Invertibility and Condition Number of the Matrix T

As mentioned above, the properties of the matrix T are crucial to determine the existence and quality of a reconstructing polynomial p^*. In particular, for a (unique) solution to the linear system 4.2 to exist, the matrix T must be invertible.

In the 1-dimensional case, the fundamental theorem of algebra guarantees that, if at least $r_{min} = 2M+1$ samples are collected, a (unique) solution p_M^* to the problem A.2 exists and can be computed solving the system A.6. In other words, the condition $r \geq 2M+1$ guarantees the matrix T to be invertible. The actual quality of the reconstruction (i.e., the actual reconstruction error), however, depends not only the number of collected samples, but also on their actual spatial distribution over the RoI. In particular, the presence of large gaps between sampling points may significantly hamper the possibility to recover the signal f with good accuracy. In this context, the condition number $k(T)$ of the matrix T allows to measure the "quality" of a given sampling geometry for the purpose of signal reconstruction.

The condition number $k(A)$ of a matrix A is defined as the ratio between the (Euclidean) norm of A and that of its inverse, thus $k(A) = \|A\|/\|A^{-1}\|$. Thus, the condition number of T is $k(T) = \|T\|_2/\|T^{-1}\|_2$. For the linear system A.6 the value of $k(T)$ gives a measure of the "sensitivity" of the solution a with respect to changes in both the sampling

geometry and the actual sampled values f_j. In particular, a low value of $k(T)$ indicates that small changes in the sampling geometry will produce only small changes in the reconstruction. This property is clearly desirable since it allows to compute a stable solution. A small condition number also makes the computed solution more robust to noise in the data. Indeed, if the sampling geometry is stable and the sampled values change only due to, for instance, measurement noise, the correspondent change in the solution is proportional to the value of $k(T)$ [86]. Thus, the smaller the condition number the less noise in the data can hamper the computation of a good reconstruction.

The consideration reported above show that the knowledge of the condition number of T allows to make some statements about the quality of a reconstruction. However, a closed analytical form of $k(T)$ is unknown and the estimation of condition numbers is usually a hard mathematical problem [75]. However, it is still possible to compute a worst-case estimate of the condition number of T by making the sampling geometry fulfill some additional constraints. In the 1-dimensional case, this consist in making the maximal distance between a sample and its closest neighbor to stay below the threshold $\delta < 1/2M$, which corresponds to the Nyquist limit in the regular sampling case [146, 171]. Thus, in the 1-dimensional case, a worst-case estimate of $k(T)$ is known if the following conditions $c1a$ and $c1b$ are fulfilled.

(c1a) The number of collected samples r is higher than $r_{min} = 2M + 1$.

(c1b) The maximal distance between adjacent samples does not exceed the limit $\delta < \frac{1}{2M}$.

We should notice that for the explicit estimate $k(T) \leq \frac{(1+2M\delta^2)}{1-2M\delta^2)}$, provided in [75], to hold, the use of the weights defined in A.3) is indispensable. Furthermore, it represents a worst-case estimate and allows to make theoretical statements about the stability of the ACT. In practical settings, however, good condition numbers can be achieved even if $c1b$ is not fulfilled over the whole segment $[0, 1]$.

In the 2-dimensional case, the geometry of the problem is more complex. A necessary condition for the matrix T to be invertible is $r \geq (2M + 1)^2$. Indeed, to determine the $(2M + 1)^2$ coefficients that generate the reconstructing polynomial p^*_M we need at least $(2M + 1)^2$ constraints to be specified. However, for a theoretical guarantee on the invertibility of T additional hypothesis are needed. To state them

here, we first report the definition of a $\delta - dense$ set from [75]. Let $D(\delta, s_j)$ be the disc of radius δ centered at $s_j = (x_j, y_j)$. The set $S_r = \{(x_j, y_j), j = 1, \ldots, r\}$ is said to be $\delta - dense$ in the unit square $[0,1]x[0,1]$, if $\bigcup_{j=1}^{r} D(\delta, s_j) \supseteq [0,1]x[0,1]$. It is easy to verify that a $\delta - dense$ set of sensor nodes provides 1-coverage of the RoI (i.e., the unit square) if the nodes are assigned a "virtual" sensing range δ. Furthermore, the distance between a sample s_j and its nearest neighbor is at most 2δ.

With this definition, we can extend conditions $c1a$ and $c1b$ to the 2-dimensional case as follows:

(c2a) The number of collected samples r is higher than $r_{min} = (2M+1)^2$.

(c2b) The sampling set $S_r = \{(x_j, y_j), j = 1, \ldots, r\}$ is $\delta - dense$ with $\delta < ln2/(4\pi M)$.

If conditions $c2a$ and $2cb$ are fulfilled, the matrix T defined as in A.9 (with weights w_j chosen as the area of the Voronoi cells of s_j) is invertible. Furthermore, the worst-case estimation of its condition number $k(T)$ is $4/(2 - e^{4\pi M\delta})$.

Besides their theoretical importance, conditions $c1b$ and $c2b$ give a practical recipe on how to select sensor nodes in a WSN so that their samples can be used to reliably reconstruct the sensor field f. As mentioned above, we can easily recognize that this requires the selected nodes to provide 1-coverage of the RoI (i.e., the segment $[0,1]$ or the unit square) assuming an "equivalent" sensing range $R_s = \delta$. Thus, coverage preserving algorithms can be used in this context as optimal sensor selection strategies.

A.4. Estimation of M

The estimates of the "equivalent" sensing range δ given by conditions $c2a$ and $c2b$ requires the value of M and, thus, of the bandwidth of the signal f to be known a-priori. A good choice of the value of M is indeed crucial to obtain a reliable solution to the reconstruction problem. Indeed, its overestimation makes the solution very sensitive to noise while its underestimation may bring to a smooth, but inaccurate reconstruction. To limit errors due to a imprecise estimation of M (and to free the user from the need to specify its value), the so-called

multi-level version of the ACT algorithm, dubbed ACT-ML, can be used [75, 169, 181].

The main rationale of the ACT-ML, is to make the reconstruction process start with an initial hypothesis on the bandwidth M, e.g. $M = 1$, and then increment its value stepwise until a given stopping criterion is satisfied. The magnitude of the stepwise increment of the bandwidth at each iteration may be 1, but also a higher value, depending on the specific case. In this way, M acts as a regularization parameter that allows to balance the stability and accuracy of the solution of the system A.6 [181]. In a set of experiments Scherzer et al. [169] also demonstrates that the ACT-ML can achieve better reconstructions using a value M far smaller that the theoretical correct value. Since the value of M determines the dimension of T, keeping it low also allows for savings in terms of computation and memory.

Therefore, given a set of samples S_r the ACT-ML can be used to perform reconstruction even if the value of M is not available. In the context of a WSN data collection scenario, however, the sampling geometry can and must be defined, and the knowledge of M is required in order to set the appropriate value of δ. In the lack of a-priori information it is still possible to define a value of δ representing a desirable spatial resolution and then refine this value is successive steps. To this scope, cross-validation techniques could be used to evaluate the quality of the reconstruction and accordingly increase or decrease the value of δ. As for the value of M, the ACT-ML will provide for its proper estimation according to the available number and distribution of the samples.

A.5. Remarks

We detailed above the importance of the value of the condition number for characterizing the performance of the ACT algorithm. We would like to note here that, in both the 1- and 2-dimensional case, the value of $k(T)$ is minimal and equal to 1 when the sampling geometry resembles a uniform grid. Therefore, given M and r, the more regular the sampling geometry is, the better the value of $k(T)$.

At the same time, the ACT performs remarkably well also in the presence of very irregular sampling geometries (mainly thanks to the use of the adaptive weights). This ability of the ACT is particularly useful in WSN scenarios, in which the actual spatial distribution of the

nodes may be unpredictable. As a comparison, the seminal approach by Duffin and Schaffer [49] is applicable only if the sampling set is a perturbation of the regular (over)sampling [74]. In general, for random irregular sampling the ACT proved to be able to outperform related approaches both in terms of computational efficiency and quality of the reconstruction [75, 181]. Furthermore, the existence of the multi-level version of the algorithm, the ACT-ML, opens the possibility to design adaptive sampling schemes able to control the number of required samples depending on the estimated value of M. Last but not least, the theoretical framework behind the ACT allows to achieve the explicit bounds expressed by conditions $c2a$ and $c2b$. Thus, the choice of the ACT as a reference reconstruction algorithm is well motivated by its practical and theoretical relevance, as well as by its wide applicability.

B. The Collection Tree Protocol (CTP)

As documented by the TinyOS TEP 119 [59], data collection is one of the fundamental primitives of a wireless sensor network (WSN). A collection protocol provides for the construction of one or more forwarding trees having each a sink as their root. Data packets are forwarded by the nodes to their parents up to at least one of the sinks. To construct the routing tree, a collection protocol maintains a function of one or more parameters upon which each node can select its parent. This function can account for the (estimated) distance in hops to the sink, the link quality, the current load of the node, its position, or residual energy. Nodes in the network can collect information about their neighboring nodes by receiving (and sending) so called routing *beacons*, which contains the information needed to evaluate the above mentioned function.

The above cited TinyOS TEP 119 specifies the requirements a collection protocol must comply with in order to work properly. For instance, it must have a mechanism to detect (and repair) routing loops. Additionally, it should be able to detect and suppress duplicates (generated as a consequence of lost acknowledgments) and to properly estimate the one-hop link quality. Different collection protocols may address these (and other) challenges using different techniques and approaches. In particular, TinyOS TEP 123 [58] describes the Collection Tree Protocol (CTP), a particular collection primitive that became quickly popular within the WSNs research community. Nonetheless, there exists an implemented version of CTP developed for the TinyOS2.x distribution and supporting several platforms (MicaZ, Telosb/TmoteSky, TinyNode).

In the following, we describe the main components of CTP and introduce some of the implementation issues that we had to face in order to implement CTP on the Castalia simulator.

CTP uses *beacon* messages (routing frames) for tree construction and maintenance, and *data* messages (data frames) to report application

data to the sink. The standard implementation of CTP reported in [58] and evaluated in [68] consists of three main logical software components: the *Link Estimator* (LE), the *Routing Engine* (RE), and the *Forwarding Engine* (FE).

The Link Estimator computes a metric to evaluate the quality of a communication link. Using this component, each node can thus assess the "goodness" of the links to its neighbors and select among them its routing parent. In CTP, the quality of a link is estimated upon statistics collected over both beacon and data frames, which converge in the so-called *Expected Transmissions* (ETX) metric. The ETX is an additive metric computed from several different values, like the fraction of lost beacons over the total sent or the decoding error at the physical layer. A throughout description of how to compute the ETX is beyond the scope of this paper and the interested reader is referred to [68]. The root of a routing tree always has an ETX equal to 0 while an internal node or leaf computes the ETX of its neighbors recursively as the sum of the ETX of the neighbor itself plus the ETX of the link to it. Given a set of links/neighbors each one with the correspondent computed ETX, CTP chooses the neighbor/link with the lowest ETX as its parent/route to the sink.

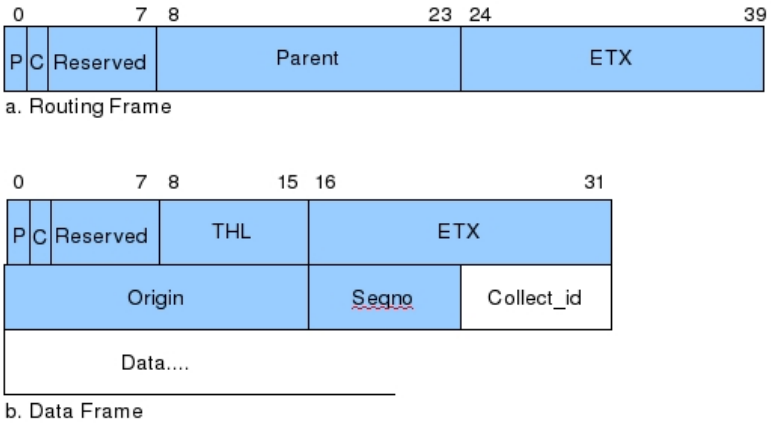


Figure B.1.: Ctp packets composition.

The Routing Engine takes care of sending, receiving and processing the routing beacons. Using the information extracted from the routing frames, the RE can maintain an updated routing table, which indicates the list of discovered neighbors, their ETX and parents. Among these neighbors, the RE can then select the parent for the current node according to the computed ETX and include this information in the routing beacons. A routing frame has a total of 5 bytes, as illustrated

in B.1a. The first two bits are the Pull (P) and Congestion (C) flags. The former is used to trigger the sending of beacon frames from neighbors for topology update, while the latter allows a node to signal that it is congested. In this case, its neighbors will look for alternatives routes for sending their packets so as to release the node from congestion. The next 6 bits are reserved for future use (e.g., additional flags). The next two bytes contain the *parent* field, which clearly indicates the identifier of the node the neighboring node sending the beacon has chosen as its parent. Next, the routing frames uses 2 bytes to report the ETX metric, which has been described above. In CTP, beacons are sent following the *Trickle* algorithm [119], i.e., the rate at which beacons are sent is progressively reduced to save communication but can be reset if specific events (such as route discovery requests) occur.

Finally, the Forwarding Engine manages data frames from the application layer and forwards incoming data packets from other nodes. The packet composition of a data frame is shown in figure B.1b. The first byte is the same as in the above described routing frame while the second byte reports the *Time Has Lived metric* (THL) metric. The THL is a counter that is incremented by one at each packet forwarding and thus indicates the number of hops a packet has effectively traveled before reaching the node handling it. The data frame further reserves two bytes for the ETX metric, and two bytes for the identifier of the node that originally sent the packet. The *SeqNo* field (1 byte) then specifies the sequence number of the current packet, as marked by the originating node. The *Collect_id* is an identifier that allows for specifying different instances of a collection service. Finally, the *data* field contains the actual payload and its length may vary according to the application requirements and considering the maximal allowed packet length. Indicating wit $N_{B_{payload}}$ the length (in bytes) of this latter field, the total length of the data frame is of $8 + N_{B_{payload}}$ bytes. Using the tuple formed by the THL, Origin, SeqNo, and Collect_id fields, the FE can detect duplicate packets. Additionally, the THL is used to allow a packet not to be discarded if a loop occurs: it will continue to be forwarded over the loop (by refreshing its THL), until a new route to the sink is found. Thus, through the action of the FE CTP also manages to suppress duplicate packets and detect routing loops. It is important to note the above described components do not work independently but closely interact through a set of well-defined interfaces.

Bibliography

[1] Kemal Akkaya and Mohamed Younis. A Survey on Routing Protocols for Wireless Sensor Networks. *Ad Hoc Networks*, 3(3):325–349, May 2005.

[2] Ian F. Akyildiz, Weiiian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless Sensor Networks: A survey. *Computer Networks*, 38(4):393–422, March 2002.

[3] Jamal N. Al-Karaki and Ahmed E. Kamal. Routing Techniques in Wireless Sensor Networks: A Survey. *IEEE Wireless Communications*, 11(6):6–28, December 2004.

[4] Javier Alberola, Ian H. Flindell, and Andrew J. Bullmore. Variability in Road Traffic Noise Levels. *Elsevier Applied Acoustic*, 66:1180–1195, April 2005.

[5] Samuel T. Alexander. *Adaptive Signal Processing: Theory and Applications*. Springer-Verlag New York Inc., New York, NY, USA, 1986.

[6] Rafael Alonso, Daniel Barbara, Hector Garcia-Molina, and Soraya Abad. Quasi-Copies: Efficient Data Sharing for Information Retrieval Systems. In *Proceedings of the International Conference on Extending Database Technology (EDBT 1988)*, pages 443–468, Venice, Italy, March 1988.

[7] Anish K. Arora, Prabal K. Dutta, Sandip S. Bapat, Vinod Kulathuman, Hongwei Zhang, Vinayak S. Naik, Vineet Mittal, Hui Cao, Murat Demirbas, Mohammed G. Gouda, Young R. Choi, Ted R. Herman, Sandeep S. Kulkarni, Umamaheswaran Arumugam, Mikhail Nesterenko, Adnan Vora, and Mark Miyashita. A Line in the Sand: A Wireless Sensor Network for Target Detection, Classification, and Tracking. *Computer Networks (Military Communications Systems and Technologies)*, 46(5):605–634, December 2004.

[8] Christopher G. Atkeson. Memory-Based Approaches to Approximating Continuous Functions. In *Proceedings of a Workshop on Nonlinear Modeling and Forecasting*, pages 503–521, Santa Fe, NM, USA, September 1990. Published by the Santa Fe Institute Studies in The Sciences of Complexity in 1992.

[9] Franz Aurenhammer. Voronoi diagrams – A Survey of a Fundamental Geometric Data Structure. *ACM Computing Surveys*, 23(3):345–405, September 1991.

[10] Aline Baggio. Wireless Sensor Networks in Precision Agriculture. In *Proceedings of the 1st Workshop on Real-World Wireless Sensor Networks (REALWSN 2005)*, Stockholm, Sweden, June 2005.

[11] Heribert Baldus, Karin Klabunde, and G. M´usch. Reliable Set-Up of Medical Body-Sensor Networks. In *Proceedings of the 1st European Workshop on Wireless Sensor Networks (EWSN 2004)*, pages 353–363, Berlin, Germany, January 2004.

[12] Guillermo Barrenetxea, Franọis Ingelrest, Gunnar Schaefer, and Martin Vetterli. The Hitchhiker's Guide to Successful Wireless Sensor Network Deployments. In *Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems (SenSys 2008)*, pages 43–56, Raleigh, NC, USA, November 2008.

[13] Can Basaran, Sebnem Baydere, Giancarlo Bongiovanni, Adam Dunkels, M. Onur Ergin, Laura Marie Feeney, Isa Hacioglu, Vlado Handziski, Andreas Köpke, Maria Lijding, Gaia Maselli, Nirvana Meratnia, Chiara Petrioli, Silvia Santini, Lodewijk van Hoesel, Thiemo Voigt, and Andrea Zanella. Embedded WiSeNts Platform Survey: Critical Evaluation of Platforms Commonly Used in Embedded Wisents Research. Available from the Embedded WiSeNts Project Website (www.embedded-wisents.org), June 2006.

[14] Boulat A. Bash, John W Byers, and Jeffrey Considine. Approximately Uniform Random Sampling in Sensor Networks. In *Proceedings of the 1st Workshop on Data Management for Sensor Networks (DMSN 2004)*, Toronto, Canada, August 2004.

[15] Boulat A. Bash and Peter J. Desnoyers. Exact Distributed Voronoi Cell Computation in Sensor Networks. In *Proceedings of the 6th International Conference on Information Processing in*

Sensor Networks (IPSN 2007), pages 236–243, Cambridge, MA, USA, April 2007.

[16] Maxim A. Batalin, Mohammad H. Rahimi, Yan Yu, Duo Liu, Aman Kansal, Gaurav S. Sukhatme, William J. Kaiser, Mark Hansen, Gregory J. Pottie, Mani Srivastava, and Deborah Estrin. Call and Response: Experiments in Sampling the Environment. In *Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys 2004)*, pages 25–38, Baltimore, MD, USA, November 2004.

[17] Lorenzo Bergamini, Carlo Crociani, and Andrea Vitaletti. Simulation vs Real Testbeds: A Validation of WSN Simulators. Technical Report 3, Sapienza Università di Roma, Dipartimento di Informatica e Sistemistica Antonio Ruberti, Rome, Italy, March 2009.

[18] Birgitta Berglund, Thomas Lindvall, and Dietrich H. Schwela. Guidelines for Community Noise. World Health organisation (WHO), 1999. Available online: `www.who.int/docstore/peh/noise/guidelines2.html`.

[19] Intel Research Lab Berkeley. Intel Lab Data. `http://db.csail.mit.edu/labdata/labdata.html`, 2004.

[20] Jan Beutel, Kay Römer, Matthias Woehrle, and Matthias Ringwald. Deployment Techniques for Wireless Sensor Networks. In *Sensor Networks: Where Theory Meets Practice*, chapter 6, pages 219–248. Springer, Heidelberg, November 2009.

[21] Shah Bhatti, James Carlson, Hui Dai, Jing Deng, Jeff Rose, Anmol Sheth, Brian Shucker, Charles Gruenwald, Adam Torgerson, and Richard Han. MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms. *Mobile Networks and Applications*, 10(4):563–579, August 2005.

[22] Edoardo Biagioni and Kent Bridges. The Application of Remote Sensor Technology to Assist the Recovery of Rare and Endangered Species. *International Journal of High Performance Computing Applications, Special Issue on Distributed Sensor Networks*, 16:315–324, August 2002.

[23] Fang Bian, David Kempe, and Ramesh Govindan. Utility-based Sensor Selection. In *Proceedings of the 5th International Conference on Information Processing in Sensor Networks (IPSN 2006)*, Nashville, TN, USA, April 2006.

[24] David A. Bies and Colin H. Hansen. *Engineering Noise Control: Theory and Practice*. Spon Press (Taylor &Francis Group), London and New York, 3rd edition, 2003.

[25] Philipp Bolliger. Redpin - Adaptive, Zero-Configuration Indoor Localization through User Collaboration. In *Proceedings of the 1st ACM International Workshop on Mobile Entity Localization and Tracking in GPS-less Environment Computing and Communication Systems*, San Francisco, CA, USA, September 2008.

[26] Athanassios Boulis. Castalia: Revealing Pitfalls in Designing Distributed Algorithms in WSN. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems (SenSys 2007)*, pages 407–408, Sydney, Australia, November 2007.

[27] Athanassios Boulis, Ansgar Fehnker, Matthias Fruth, and Annabelle McIver. CaVi – Simulation and Model Checking for-Wireless Sensor Networks. In *Proceedings of the 5th International Conference on Quantitative Evaluation of Systems (QEST 2008)*, pages 37–38, Saint Malo, France, September 2008.

[28] G.E.P. Box and G.M. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day Series in Time Series Analysis. Holden-Day, San Francisco, CA, USA, 1976.

[29] Nirupama Bulusu, John Heidemann, and Deborah Estrin. GPS-less Low-cost Outdoor Localization for Very Small Devices. *IEEE Personal Communications*, 7(5):28–34, October 2000.

[30] Phil Buonadonna, David Gay, Joseph M. Hellerstein, Wei Hong, and Samuel Madden. TASK: Sensor Network in a Box. In *Proceedings of the 2nd IEEE European Workshop on Wireless Sensor Networks and Applications (EWSN 2005)*, Istanbul, Turkey, February 2005.

[31] Nicolas Burri, Pascal von Rickenbach, and Roger Wattenhofer. Dozer: Ultra-Low Power Data Gathering in Sensor Networks. In *Proceedings of the 6th International Conference on Information*

Processing in Sensor Networks (IPSN 2007), Cambridge, MA, USA, April 2007.

[32] Hans J. Butterweck. A Wave Theory of Long Adaptive Filters. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 48(6):739–747, June 2001.

[33] John Byers and Gabriel Nasser. Utility-based Decision-making in Wireless Sensor Networks. Technical Report CS-TR-2000-014, Computer Science Department, Boston University, October 2000.

[34] John Byers and Gabriel Nasser. Utility-based Decision-making in Wireless Sensor Networks. In *Proceedings of the 1st ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc 2000)*, pages 143–144, 2000.

[35] CALM Network. Blue Book 2006 – Research on Environmental Noise, April 2006. `www.calm-network.com/bluebook/content/intro.htm`.

[36] CALM Network. Research for a Quieter Europe in 2020 – Updated Strategy Paper of the CALM Network, September 2007. `www.calm-network.com/index_stratpap.htm`.

[37] Alberto Camilli, Carlos E. Cugnasca, Antonio M. Saraiva, André R. Hirakawa, and Pedro L. P. Corrêa. From Wireless Sensors to Field Mapping: Anatomy of an Application for Precision Agriculture. *Computers and Electronics in Agriculture*, 58(1):25–36, August 2007.

[38] Andrew T. Campbell, Shane B. Eisenman, Nicholas D. Lane, Emiliano Miluzzo, and Ronald A. Peterson. People-Centric Urban Sensing. In *Proceedings of the 2nd Annual International Wireless Internet Conference (WICON 2006)*, Boston, MA, USA, August 2006.

[39] Qing Cao, Tarek Abdelzaher, Tian He, and John Stankovic. Towards Optimal Sleep Scheduling in Sensor Networks for Rare-Event Detection. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN 2005)*, Los Angeles, CA, USA, April 2005.

[40] Wook Choi and Sajal K. Das. Coverage-Adaptive Random Sensor Scheduling for Application-Aware Data Gathering in Wireless Sensor Networks. *Elsevier Computer Communications*, 29(17):3467–3482, November 2006.

[41] Cirrus Research Plc. `www.cirrusresearch.co.uk`.

[42] Robert L. Cook. Stochastic Sampling in Computer Graphics. *ACM Transactions on Graphics (TOG)*, 5(1):51–72, January 1986.

[43] Crossbow Technology Inc. `www.xbow.com`.

[44] Abhimanyu Das and David Kempe. Sensor Selection for Minimizing Worst-case Prediction Error. In *Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN 2008)*, pages 97–108, St. Louis, MO, USA, April 2008.

[45] Jan D. De Gooijer and Rob J. Hyndman. 25 Years of Time Series Forecasting. *Elsevier International Journal of Forecasting*, 22(3):443–473, 2006.

[46] Amol Deshpande, Carlos Guestrin, Samuel R. Madden, Joseph M. Hellerstein, and Wei Hong. Model-Driven Data Acquisition in Sensor Networks. In *Proceedings of the 30th Very Large Data Base Conference (VLDB 2004)*, Toronto, Canada, August 2004.

[47] Jiagen (Jason) Ding, Sing-Yiu Cheung, Chin-Woo Tan, and Pravin Varaiya. Signal Processing of Sensor Node Data for Vehicle Detection. In *Proceedings of the 7th IEEE International Conference on Intelligent Transportation Systems*, Washington D.C., USA, October 2004.

[48] Min Dong, Lang Tong, and Brian M. Sadler. Impact of Data Retrieval Pattern on Homogeneous Signal Field Reconstruction in Dense Networks. *IEEE Transactions on Signal Processing*, 54(11):4352–4364, November 2006.

[49] R.J. Duffin and A.C. Schaeffer. A Class of Nonharmonic Fourier Series. *Transactions of the American Mathematical Society*, 72(72):341–366, March 1952.

[50] Adam Dunkels, Björn Grönvall, and Thiemo Voigt. Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proceedings of the Annual IEEE Conference on Local Computer Networks (LCN 2004)*, pages 455–462, Los Alamitos, CA, USA, November 2004.

[51] Shane B. Eisenman, Emiliano Miluzzo, Nicholas D. Lane, Ronald A. Peterson, Gahng-Seop Ahn, and Andrew T. Campbell. The BikeNet Mobile Sensing System for Cyclist Experience Mapping. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems (SenSys 2007)*, pages 87–101, Sydney, Australia, November 2007.

[52] Deborah Estrin, Ramesh Govindan, John Heidemann, and Satish Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. In *Proceedings of the 5th annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 1999)*, pages 263–270, Seattle, WA, USA, August 1999.

[53] European Commission. Green Paper on Future Noise Policy. `http://ec.europa.eu/environment/noise/greenpap.htm`, November 1996.

[54] European Commission Working Group Assessment of Exposure to Noise (WG-AEN). Good Practice Guide for Strategic Noise Mapping and the Production of Associated Data on Noise Exposure. Position Paper (Final Draft). `http://ec.europa.eu/environment/noise/pdf/wg_aen.pdf`, January 2006.

[55] Extech Instruments Corporation. Digital Sound Level Meter Model 407740 - User's Guide. `www.extech.com/instruments/resources/manuals/407740_UM.pdf`. Version 1.6 08/05.

[56] Hans G. Feichtinger, Karlheinz Gröchenig, and Thomas Strohmer. Efficient Numerical Methods in Non-Uniform Sampling Theory. *Numerische Mathematik*, 69(4):423–440, February 1995.

[57] Luca Filipponi, Silvia Santini, and Andrea Vitaletti. Data Collection in Wireless Sensor Networks for Noise Pollution Monitoring. In *Proceedings of the 4th IEEE/ACM International Conference on Distributed Computing in Sensor Systems (DCOSS 2008), Santorini Island, Greece, June*, 2009.

[58] Rodrigo Fonseca, Omprakash Gnawali, Kyle Jamieson, Sukun Kim, Philip Levis, and Alec Woo. TinyOS Enhancement Proposal (TEP) 123: The Collection Tree Protocol (CTP). `www.tinyos.net/tinyos-2.x/doc/pdf/tep123.pdf`.

[59] Rodrigo Fonseca, Omprakash Gnawali, Kyle Jamieson, and Philip Levis. TinyOS Enhancement Proposal (TEP) 119: Collection. `www.tinyos.net/tinyos-2.x/doc/pdf/tep119.pdf`.

[60] Christian Frank, Philipp Bolliger, Friedemann Mattern, and Wolfgang Kellerer. The Sensor Internet at Work: Locating Everyday Items Using Mobile Phones. *Pervasive and Mobile Computing*, 4(3):421–447, June 2008.

[61] Christian Frank and Kay Römer. Algorithms for Generic Role Assignment in Wireless Sensor Networks. In *Proceedings of the 3rd ACM Conference on Embedded Networked Sensor Systems (SenSys 2005)*, San Diego, CA, USA, November 2005.

[62] Deepak Ganesan, Ben Greenstein, Denis Perelyubskiy, Deborah Estrin, and John Heidemann. An Evaluation of Multi-Resolution Storage for Sensor Networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys 2003)*, pages 89–102, Los Angeles, CA, USA, November 2003.

[63] Deepak Ganesan, Sylvia Ratnasamy, Hanbiao Wang, and Deborah Estrin. Coping with Irregular Spatio-Temporal Sampling in Sensor Networks. *ACM SIGCOMM Computer Communication Review*, 34(1):125–130, January 2004. This article has also been published in the Proceedings of the 2nd Workshop on Hot Topics in Networks (HotNets-II), Cambridge, MA, USA, November 2003.

[64] Everette S. Gardner. Exponential Smoothing: The State of the Art. *Journal of Forecasting*, 4:1–38, 1985.

[65] Everette S. Gardner. Exponential Smoothing: The State of the Art – Part II. *International Journal of Forecasting*, 22(4):637–666, October–December 2006.

[66] F. Garwood. The Variance of the Overlap of Geometrical Figures with Reference to a Bombing Problem. *Biometrika*, 34(1/2):1–17, January 1947.

[67] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesC Language: A Holistic Approach to Networked Embedded Systems. *SIGPLAN Notice*, 38(5):1–11, May 2003. This Article has also been published in the Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2003), San Diego, CA, USA, June 2003.

[68] Omprakash Gnawali, Rodrigo Fonseca, Kyle Jamieson, and Philip Levis. CTP: Robust and Efficient Collection through Control and Data Plane Integration. Technical report, The Stanford Information Networks Group (SING), August 2008. `http://sing.stanford.edu/pubs/sing-08-02.pdf`.

[69] Samir Goel and Tomasz Imielinski. Prediction-Based Monitoring in Sensor Networks: Taking Lessons from MPEG. *ACM SIGCOMM Computer Communication Review Special Issue on Wireless Extensions to the Internet*, 31(5):82–98, October 2001.

[70] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins Studies in Mathematical Sciences. Johns Hopkins University Press, London / Baltimore, 3rd edition, 1996.

[71] GoodFood European Integrated Project. Sensor Network in a Vineyard – Food Safety and Quality Monitoring with Microsystems. `www3.unifi.it/midra/goodfood/`.

[72] Ben Greenstein, Eddie Kohler, and Deborah Estrin. A Sensor Network Application Construction Kit (SNACK). In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys 2004)*, Baltimore, MD, USA, November 2004.

[73] Ben Greenstein, Christopher Mar, Alex Pesterev, Shahin Farshchi, Eddie Kohler, Jack Judy, and Deborah Estrin. Capturing High-Frequency Phenomena Using a Bandwidth-Limited Sensor Networks. In *Proceedings of the 4th ACM International Conference on Embedded Networked Sensor Systems (SenSys 2006)*, Boulder, CO, USA, November 2006.

[74] Karlheinz Gröchenig. Reconstruction Algorithms in Irregular Sampling. *Mathematics of Computation*, 59(199):181–194, July 1992.

[75] Karlheinz Gröchenig and Thomas Strohmer. *Nonuniform Sampling: Theory and Practice*, chapter Numerical and Theoretical Aspects of Non-Uniform Sampling of Band-Limited Images. Information Technology: Transmission, Processing and Storage. Springer, Berlin / Heidelberg, 2001.

[76] Carlos Guestrin, Peter Bodik, Romain Thibaux, Mark Paskin, and Samuel Madden. Distributed Regression: An Efficient Framework for Modeling Sensor Network Data. In *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks (IPSN 2004)*, pages 1–10, Berkeley, CA, USA, April 2004.

[77] Piyush Gupta and P.R. Kumar. The Capacity of Wireless Networks. *IEEE Transactions on Information Theory*, 46(2):388–404, March 2000.

[78] Richard Guy, Ben Greenstein, John Hicks, Rahul Kapur, Nithya Ramanathan, Tom Schoellhammer, Thanos Stathopoulos, Karen Weeks, Kevin Chang, Lew Girod, and Deborah Estrin. Experiences with the Extensible Sensing System ESS. Technical Report 01-310-825-3127, UCLA Center for Embedded Network Sensing, January 2006.

[79] Peter Hall. *Introduction to the Theory of Coverage Processes*. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, New York Chichester Brisbane Toronto Singapore, October 1988.

[80] James D. Hamilton. *Time Series Analysis*. Princeton University Press, Princeton, NJ, USA, 1994.

[81] Jarvis Haupt, Waheed U. Bajwa, Michael Rabbat, and Robert Nowak. Compressed Sensing for Networked Data – A Different Approach to Decentralized Compression. *IEEE Signal Processing Magazine*, 25(2):92–101, March 2008.

[82] Simon Haykin. *Adaptive Filter Theory*. Prentice Hall Information And System Sciences Series. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 4th edition, 2004.

[83] Simon Haykin and Bernard Widrow, editors. *Least-Mean-Square Adaptive Filters*. Wiley Series on Adaptive and Learning Systems

for Signal Processing, Communication and Control. John Willey & Sons, Inc., New York, 2003.

[84] Wendi R. Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. An Application-Specific Protocol Architecture for Wireless Microsensor Networks. *IEEE Transactions on Wireless communications*, 1(4):660–670, October 2002.

[85] Wendi R. Heinzelman, Joanna Kulik, and Hari Balakrishnan. Adaptive Protocols for Information Dissemination in Wireless Sensor Networks. In *Proceedings of the 5th annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 1999)*, pages 174–185, Seattle, WA, United States. August, 1999.

[86] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, USA, 2nd edition, 2002.

[87] Wassily Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 58(301):13–30, March 1963.

[88] Hans Huber and Fridolin Keller. Personal communication, March 2007.

[89] Jonathan Hui, Philip Levis, and David Moss. TinyOS Enhancement Proposal (TEP) 125: TinyOS 802.15.4 Frames. `www.tinyos.net/tinyos-2.x/doc/html/tep125`.

[90] Rob J. Hyndman, Anne B. Koehler, Ralph D. Snyder, and Simone Grose. A State Space Framework for Automatic Forecasting Using Exponential Smoothing Methods. *International Journal of Forecasting*, 18(3):439–454, September 2002.

[91] International Electrotechnical Commission. Electroacoustics – Sound Level Meters – Part 1: Specifications, May 2002.

[92] International Organization for Standardization (ISO). Acoustics – Description, Measurement and Assessment of Environmental Noise – Part 1: Basic Quantities and Assessment Procedures, August 2003.

[93] International Organization for Standardization (ISO). Acoustics – Description, Measurement and Assessment of Environmental Noise – Part 2: Determination of Environmental Noise Levels, March 2007.

[94] Prakash Ishwar, Rohit Puri, S. Sandeep Pradhan, and Kannan Ramchandran. Distributed Sampling for Dense Sensor Networks: A "Bit-Conservation Principle". In *Proceedings of the 2nd International Workshop on Information Processing in Sensor Networks (IPSN 2003)*, pages 17–31, Palo Alto, CA, USA, April 2003.

[95] Prakash Ishwar, Rohit Puri, S. Sandeep Pradhan, and Kannan Ramchandran. On Rate-Constrained Distributed Estimation in Unreliable Sensor Networks. *IEEE Journal on Selected Areas in Communications*, 23(4):765–775, April 2005.

[96] Ranjit Iyer and Leonard Kleinrock. QoS Control for Sensor Networks. In *Proceedings of the IEEE International Conference on Communications (ICC 2003)*, Anchorage, AK, USA, May 2003.

[97] 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, Paris, France, June 2004.

[98] Siddharth Joshi and Stephen Boyd. Sensor Selection via Convex Optimization. *IEEE Transactions on Signal Processing*, 57(2):451–462, February 2009.

[99] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Peh Peh, Li-Shiuan, and Daniel Rubenstein. Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2002)*, pages 96–107, San Jose, CA, USA, October 2002.

[100] Joseph M. Kahn, Randy H. Katz, and Kristofer S. J. Pister. Next Century Challenges: Mobile Networking for "Smart Dust". In *Proceedings of the 5th annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 1999)*, pages 271–278, Seattle, WA, USA, August 1999.

[101] Rudolf E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME, Journal of Basic Engineering, Series D*, 82:35–45, 1960.

[102] Eiman Kanjo, Steve Benford, Mark Paxton, Alan Chamberlain, Danae Stanton Fraser, Dawn Woodgate, David Crellin, and Adrain Woolard. MobGeoSen: Facilitating Personal GeoSensor Data Collection and Visualization using Mobile Phones. *Personal Ubiquitous Computing Journal*, 12(8):599–607, November 2008.

[103] Cornelia Kappler and Georg Riegel. A Real-World, Simple Wireless Sensor Network for Monitoring Electrical Energy Consumption. In *Proceedings of the 1st European Workshop on Wireless Sensor Networks (EWSN 2004)*, pages 339–352, Berlin, Germany, January 2004.

[104] Holger Karl and Andreas Willig. *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England, 2005.

[105] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, Englewood Cliffs, NJ, USA, 2nd edition, March 1988.

[106] Sukun Kim, Shamim Pakzad, David Culler, James Demmel, Gregory Fenves, Steven Glaser, and Martin Turon. Health Monitoring of Civil Infrastructures Using Wireless Sensor Networks. In *Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN 2007)*, pages 254–263, Cambridge, MA, USA, April 2007.

[107] David Kotz, Calvin Newport, and Chip Elliott. The Mistaken Axioms of Wireless-Network Research. Technical Report TR 2003-467, Dartmouth College Computer Science, July 2003.

[108] Philipp Küderli. Adaptive Model Selection Library for TinyOS. Master's thesis, ETH Zurich, May 2008.

[109] Fabian Kuhn, Thomas Locher, and Roger Wattenhofer. Distributed Selection: a Missing Piece of Data Aggregation. *Communications of the ACM*, 51(9):93–99, September 2008.

[110] Santosh Kumar, Ten H. Laiand, and József Balogh. On k-Coverage in a Mostly Sleeping Sensor Network. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking (MobiCom 2004)*, pages 144–158, Philadelphia, PA, USA, September 2004.

[111] Santosh Kumar, Ten H. Laiand, and József Balogh. On k-Coverage in a Mostly Sleeping Sensor Network. *Wireless Networks*, 14(3):277–294, June 2008.

[112] Nicholas D. Lane, Shane B. Eisenman, Mirco Musolesi, Emiliano Miluzzo, and Andrew T. Campbell. Urban Sensing Systems: Opportunistic or Participatory? In *Proceedings of the 9th workshop on Mobile Computing Systems and Applications (HotMobile 2008)*, pages 11–16, Napa Valley, CA, USA, February 2008.

[113] Koen Langendoen and Niels Reijers. Distributed Localization in Wireless Sensor Networks: A Quantitative Comparison. *Computer Networks*, 43(4):499–518, November 2003.

[114] Iosif Lazaridis and Sharad Mehrotra. Capturing Sensor-Generated Time Series with Quality Guarantee. In *Proceedings of the 19th International Conference on Data Engineering (ICDE 2003)*, pages 429–440, Bangalore, India, March 2003.

[115] Yann-Aël Le Borgne and Gianluca Bontempi. Round Robin Cycle for Predictions in Wireless Sensor Networks. In *2nd International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP 2005)*, pages 253–258, Melbourne, Australia, December 2005.

[116] Yann-Aël Le Borgne, Silvia Santini, and Gianluca Bontempi. Adaptive Model Selection for Time Series Prediction in Wireless Sensor Networks. *International Journal for Signal Processing, Special Issue on Information Processing and Data Management in Wireless Sensor Networks*, 87(12):3010–3020, December 2007.

[117] Didier Le Gall. MPEG: A Video Compression Standard for Multimedia Applications. *Communications of the ACM, Special Issue on Digital Multimedia Systems*, 34(4):46–58, April 1991.

[118] Philip Levis, Samuel 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. In *Ambient Intelligence*, chapter 2, pages 115–148. Springer, Berlin Heidelberg, December 2005.

[119] Philip Levis, Neil Patel, David Culler, and Scott Shenker. A Self-Regulating Algorithm for Code Maintenance and Propagation in Wireless Sensor Networks. In *Proceedings of the 1st USENIX Conference on Networked Systems Design and Implementation (NSDI 2004)*, pages 15–28, San Francisco, CA, USA, March 2004.

[120] Song Lin, Benjamin Arai, Dimitrios Gunopulos, and Gautam Das. Region Sampling: Continuous Adaptive Sampling on Sensor Networks. In *Proceedings of the 24th IEEE International Conference of Data Engineering (ICDE 2008)*, pages 794–803, Cancun, Mexico, April 2008.

[121] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless Sensor Networks for Habitat Monitoring. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA 2002)*, pages 88–97, Atlanta, GA, USA, September 2002.

[122] Spyros Makridakis, Steven C. Wheelwright, and Rob J. Hyndman. *Forecasting: Methods and Applications.* John Wiley & Sons, Inc., 3rd edition, 1998.

[123] Daniel Marco, Enrique J. Duarte-Melo, Mingyan Liu, and David L. Neuhoff. On the Many-to-One Transport Capacity of a Dense Wireless Sensor Network and the Compressibility of Its Data. In *Proceedings of the 2nd International Workshop on Information Processing in Sensor Networks (IPSN 2003)*, pages 1–16, Palo Alto, CA, USA, April 2003.

[124] Oden Maron and Andrew W. Moore. The Racing Algorithm: Model Selection for Lazy Learners. *Artificial Intelligence Review*, 11(1-5):193–225, February 1997.

[125] Fernando Martincic and Loren Schwiebert. Introduction to Wireless Sensor Networking. In *Handbook of Sensor Networks: Algorithms and Architectures*, chapter 1, pages 1–40. John Willey & Sons, Inc., Hoboken, NJ, USA, 2005.

[126] Farokh Marvasti, editor. *Nonuniform Sampling: Theory and Practice*. Information Technology: Transmission, Processing and Storage. Springer, Berlin / Heidelberg, 2001.

[127] Pina Marziliano and Martin Vetterli. Reconstruction of Irregularly Sampled Discrete-Time Bandlimited Signals with Unknown Sampling Locations. *IEEE Transactions on Signal Processing*, 48(12):3462–3471, December 2000.

[128] Gaurav Mathur, Peter Desnoyers, Deepak Ganesan, and Prashant Shenoy. Ultra-Low Power Data Storage for Sensor Networks. In *Proceedings of the 5th International Conference on Information Processing in Sensor Networks (IPSN 2006)*, pages 374–381, Nashville, TN, USA, April 2006.

[129] William M. Merrill, Fredric Newberg, Kathy Sohrabi, William J. Kaiser, and Gregory J.Rr Pottie. Collaborative Networking Requirements for Unattended Ground Sensor Systems. In *Proceedings of the IEEE Aerospace Conference*, volume 5, pages 2153–2165, Big Sky, MT, USA, March 2003.

[130] Emiliano Miluzzo, Nicholas D. Lane, Kristóf Fodor, Ronald Peterson, Hong Lu, Mirco Musolesi, Shane B. Eisenman, Xiao Zheng, and Andrew T. Campbell. Sensing Meets Mobile Social Networks: the Design, Implementation and Evaluation of the CenceMe Application. In *Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems (SenSys 2008)*, pages 337–350, Raleigh, NC, USA, November 2008.

[131] Ananya Misra, Georg Essl, and Michael Rohs. Microphone as Sensor in Mobile Phone Performance. In *Proceedings of the 8th International Conference on New Interfaces for Musical Expression (NIME 2008)*, Genova, Italy, June 2008.

[132] Don P. Mitchell. Generating Antialiased Images at Low Sampling Densities. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 1987)*, pages 65–72, Anaheim, CA, USA, July 1987.

[133] Prashanth Mohan, Venkata N. Padmanabhan, and Ramachandran Ramjee. Nericell: Rich Monitoring of Road and Traffic Conditions Using Mobile Smartphones. In *Proceedings of the 6th*

ACM Conference on Embedded Networked Sensor Systems (SenSys 2008), pages 323–336, Raleigh, NC, USA, November 2008.

[134] Douglas C. Montgomery, Lynwood A. Johnson, and John S. Gardiner. *Forecasting and Time Series Analysis.* McGraw-Hill, New York, NY, USA, 2nd edition, 1990.

[135] George Moschytz and Markus Hofbauer. *Adaptive Filters.* Springer Verlag, Berlin, 2000.

[136] Moteiv Corporation. Accelerating Sensor Networking. `www.moteiv.com` (`www.sentilla.com`).

[137] Moteiv Corporation. Moteiv Hardware Product Transition - Tmote Sky and Tmote invent documentation. `www.sentilla.com/moteiv-transition.html`.

[138] Rohan Murty, Geoffrey Mainland, Ian Rose, Atanu Roy Chowdhury, Abhimanyu Gosain, Josh Bers, and Matt Welsh. CitySense: A Vision for an Urban-scale Wireless Networking Testbed. In *Proceedings of the IEEE International Conference on Technologies for Homeland Security*, Waltham, MA, USA, May 2008.

[139] National Oceanic Atmospheric Administartion's National Data Buoy Center. Center of Excellence in Marine Technology. Historical Data Repository: `www.ndbc.noaa.gov/historical_data.shtml`.

[140] Alessandro Nordio, Carla-Fabian Chiasserini, and Emanuele Viterbo. Bandlimited Field Reconstruction for Wireless Sensor Networks. *ArXiv e-prints*, July 2007. `http://adsabs.harvard.edu/abs/2007arXiv0707.1954N`.

[141] Alessandro Nordio, Carla-Fabiana Chiasserini, and Emanuele Viterbo. Quality of Field Reconstruction in Sensor Networks. In *Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM 2007)*, pages 2406–2410, Anchorage , AK , USA, May 2007.

[142] Alessandro Nordio, Carla-Fabiana Chiasserini, and Emanuele Viterbo. Performance of Linear Reconstruction Techniques With Noise and Uncertain Sensor Locations. *IEEE Transactions on Signal Processing*, 56(8):3535–3547, August 2008.

[143] Alessandro Nordio, Carla-Fabiana Chiasserini, and Emanuele Viterbo. Signal Reconstruction in Multidimensional Sensor Fields. In *International Zurich Seminar on Communications (IZS 2008)*, Zurich, Switzerland, March 2008.

[144] Chris Olston, Jing Jiang, and Jennifer Widom. Adaptive Filters for Continuous Queries over Distributed Data Streams. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 563–574, San Diego, CA, USA, June 2003.

[145] Chris Olston and Jennifer Widom. Best-Effort Cache Synchronization with Source Cooperation. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 73–84, Madison, WI, USA, June 2002.

[146] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[147] Luigi Palopoli, Roberto Passerone, Amy L. Murphy, Gian Pietro Picco, and Alessandro Giusti. Solving the Wake-up Scattering Problem Optimally. In *Proceedings of the 6th European Conference on Wireless Sensor Networks (EWSN 2009)*, pages 166–182, Cork, Ireland, February 2009.

[148] Sung Park, Andreas Savvides, and Mani B. Srivastava. SensorSim: a Simulation Framework for Sensor Networks. In *Proceedings of the 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM 2000)*, pages 104–111, Boston, MA, USA, August 2000.

[149] Neal Patwari, Joshua N. Ash, Spyros Kyperountas, Alfred O. III Hero, Randolph L. Moses, and Neiyer S. Correal. Locating the Nodes: Cooperative Localization in Wireless Sensor Networks. *IEEE Signal Processing Magazine*, 22(4):54–69, July 2005.

[150] Mark Perillo and Wendi R. Heinzelman. DAPR: A Protocol for Wireless Sensor Networks Utilizing an Application-based Routing Cost. In *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC 2004)*, pages 1528–1533, Atlanta, GA, USA, March 2004.

[151] Mark Perillo and Wendi R. Heinzelman. An Integrated Approach to Sensor Role Selection. *IEEE Transactions on Mobile Computing*, 8(5):709–720, May 2009.

[152] Mark Perillo, Zeljko Ignjatovic, and Wendi R. Heinzelman. An Energy Conservation Method for Wireless Sensor Networks Employing a Blue Noise Spatial Sampling Technique. In *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks (IPSN 2004)*, pages 116–123, Berkeley, CA, USA, April 2004.

[153] Hai N. Pham, Dimosthenis Pediaditakis, and Athanassios Boulis. From Simulation to Real Deployments in WSN and Back. In *Proceedings of the IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2007)*, pages 1–6, Helsinki, Finland, June 2007.

[154] Joseph Polastre, Robert Szewczyk, and David Culler. Telos: Enabling Ultra-Low Power Wireless Research. In *Proceedings of the 4th International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS 2005)*, pages 364–369, Los Angeles, CA, USA, April 2005.

[155] S. Sandeep Pradhan, Julius Kusuma, and Kannan Ramchandran. Distributed Compression in a Dense Microsensor Network. *IEEE Signal Processing Magazine*, 19(2):51–60, March 2002.

[156] S. Sandeep Pradhan and Kannan Ramchandran. Distributed Source Coding Using Syndromes (DISCUS): Design and Construction. In *Proceedings of the 1999 Data Compression Conference (DCC 1999)*, pages 158–167, Snowbird, UT, USA, March 1999.

[157] Mohammad Rahimi, Richard Pon, William J. Kaiser, Gaurav S. Sukhatme, Deborah Estrin, and Mani Srivastava. Adaptive Sampling for Environmental Robotics. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2004)*, volume 4, pages 3537–3544, New Orleans, LA, USA, April 2004.

[158] Michael Rauth and Thomas Strohmer. Smooth Approximation of Potential Fields from Noisy Scattered Data. *Geophysics*, 63(1):85–94, January-February 1998.

[159] Kay Römer, Philipp Blum, and Lennart Meier. Time Synchronization and Calibration in Wireless Sensor Networks. In *Handbook of Sensor Networks: Algorithms and Architectures*, Wiley Series on Parallel and Distributed Computing, chapter 7, pages 199–237. John Wiley & Sons, Inc., 2005.

[160] Kay Römer and Friedemann Mattern. The Design Space of Wireless Sensor Networks. *IEEE Wireless Communications*, 11(6):54–61, December 2004.

[161] Paolo Santi. Topology Control in Wireless Ad Hoc and Sensor Networks. *ACM Computing Surveys*, 37(2):164–194, June 2005.

[162] Silvia Santini. TinyLAB: A Matlab-Based Framework for Interaction with Wireless Sensor Networks. In *1st European TinyOS Technology Exchange (ETTX 2009)*, Cork, Ireland, February 2009.

[163] Silvia Santini and Ugo Colesanti. Adaptive Random Sensor Selection for Field Reconstruction in Wireless Sensor Networks. In *Proceedings of the 6th International Workshop on Data Management for Sensor Networks (DMSN 2009)*, Lyon, France, August 2009.

[164] Silvia Santini, Benedikt Ostermaier, and Robert Adelmann. On the Use of Sensor Nodes and Mobile Phones for the Assessment of Noise Pollution Levels in Urban Environments. In *Proceedings of the Sixth International Conference on Networked Sensing Systems (INSS 2009)*, Pittsburgh, PA, USA, June 2009.

[165] Silvia Santini, Benedikt Ostermaier, and Andrea Vitaletti. First Experiences Using Wireless Sensor Networks for Noise Pollution Monitoring. In *Proceedings of the Third ACM Workshop on Real-World Wireless Sensor Networks (REALWSN 2008)*, pages 61–65, Glasgow, United Kingdom, April 2008.

[166] Silvia Santini and Kay Römer. An Adaptive Strategy for Quality-Based Data Reduction in Wireless Sensor Networks. In *Proceed-*

ings of the 3rd International Conference on Networked Sensing Systems (INSS 2006), Chicago, IL, USA, June 2006.

[167] Silvia Santini and Andrea Vitaletti. Wireless Sensor Networks for Environmental Noise Monitoring. 6. GI/ITG KuVS Fachgespräch "Drahtlose Sensornetze", Aachen, Germany, July 2007.

[168] Anna Scaglione and Sergio D. Servetto. On the Interdependence of Routing and Data Compression in Multi-Hop Sensor Networks. *Wireless Networks*, 11(1-2):149–160, January 2005.

[169] Otmar Scherzer and Thomas Strohmer. A Multi-Level Algorithm for the Solution of Moment Problems. *Numerical Functional Analysis and Optimization*, 19(3&4):353–375, 1998.

[170] Sergio D. Servetto. Sensing Lena – Massively Distributed Compression of Sensor Images. In *Proceedings of the 2003 International Conference on Image Processing (ICIP 2003)*, pages 613–616, Barcelona, Spain, September 2003. Special Session on Distributed Source Coding (Invited Paper).

[171] Claude E. Shannon. Communication in the Presence of Noise. *Proceedings of the Institute of Radio Engineers*, 37(1):10–21, January 1949.

[172] Donald Shepard. A Two-Dimensional Interpolation Function for Irregularly-Spaced Data. In *Proceedings of the 23rd ACM National Conference/Annual Meeting*, pages 517–524, August 1968.

[173] Shockfish SA. TinyNode 184 Embedded Wireless Network Node – Fact Sheet, November 2008. `www.tinynode.com/uploads/media/SH-TN184-103_rev1.1.pdf`.

[174] Brian Shucker, Jeff Rose, Anmol Sheth, James Carlson, Shah Bhatti, Hui Dai, Jing Deng, and Richard Han. Embedded Operating Systems for Wireless Microsensor Nodes. In *Handbook of Sensor Networks: Algorithms and Architectures*, Wiley Series on Parallel and Distributed Computing, chapter 6. John Willey & Sons, Inc., 2005.

[175] Adam Silberstein, Rebecca Braynard, and Jun Yang. Constraint Chaining: On EnergyŋEfficient Continuous Monitoring in Sensor Networks. In *Proceedings of the 22nd ACM SIGMOD In-*

ternational Conference on Management of Data, pages 157–168, Chicago, IL, USA, June 2006.

[176] Gyula Simon, Miklós Maróti, Ákos Lédeczi, György Balogh, Branislav Kusy, András Nádas, Gábor Pap, János Sallai, and Ken Frampton. Sensor Network-Based Countersniper System. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys 2004)*, pages 1–12, Baltimore, MD, USA, November 2004.

[177] Mani Srivastava, Mark Hansen, Jeff Burke, Andrew Parker, Sasank Reddy, Ganeriwal Saurabh, Mark Allman, Vern Paxson, and Deborah Estrin. Wireless Urban Sensing Systems. Technical Report 65, Center for Embedded Networked Sensing Systems (CENS), University of California Los Angeles (UCLA), April 2006.

[178] Anders Stenman, Fredrik Gustafsson, and Lennart Ljung. Just in Time Models for Dynamical Systems. In *Proceedings of the 35th IEEE Conference on Decision and Control*, pages 1115–1120, Kobe, Japan, 1996.

[179] Ivan Stojmenovic, editor. *Handbook of Sensor Networks: Algorithms and Architectures*. Wiley Series on Parallel and Distributed Computing. John Willey & Sons, Inc., Hoboken, NJ, USA, 2005.

[180] Thomas Strohmer. Computationally Attractive Reconstruction of Band-Limited Images from Irregular Samples. *IEEE Transactions on Image Processing*, 6(4):540–548, April 1997.

[181] Thomas Strohmer. Numerical Analysis of the Non-Uniform Sampling Problem. *Journal of Computational and Applied Mathematics, Special Issue on Numerical Analysis (Vol. II: Interpolation and Extrapolation)*, 122(1-2):297–316, October 2000.

[182] Thomas Strohmer, Thomas Binder, and Michael Süssner. How to Recover Smooth Object Boundaries in Noisy Medical Images. In *Proceedings of the International Conference on Image Processing (ICIP 1996)*, pages 331–334, Lausanne, Switzerland, September 1996.

[183] Jun-Zhao Sun and Jaakko Sauvola. Cross-Layer Optimization Framework for Wireless Sensor Networks. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2006)*, pages 2029–2034, Beijing, China, October 2006.

[184] Norman R. Swanson and Halbert White. A Model Selection Approach to Real-Time Macroeconomic Forecasting Using Linear Models and Artificial Neural Networks. *The Review of Economics and Statistics*, 79(4):540–550, November 1997.

[185] Robert Szewczyk, Alan Mainwaring, Joseph Polastre, John Anderson, and David Culler. An Analysis of a Large Scale Habitat Monitoring Application. In *Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys 2004)*, pages 214–226, Baltimore, MD, USA, November 2004.

[186] Igor Talzi, Andreas Hasler, Stephan Gruber, and Christian Tschudin. PermaSense: Investigating Permafrost with a WSN in the Swiss Alps. In *Proceedings of the 4th ACM Workshop on Embedded Networked Sensors (EmNets 2007)*, pages 8–12, Cork, Ireland, June 2007.

[187] The European Parlament and the Council. Directive 2002/49/EC of the European Parliament and of the Council of 25 June 2002 relating to the Assessment and Management of Environmental Noise. Official Journal of the European Communities, July 2002.

[188] Di Tian and Nicolas D. Georganas. A Coverage-Preserving Node Scheduling Scheme for Large Wireless Sensor Networks. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, pages 32–41, Atlanta, GA, USA, September 2002.

[189] TinyOS. An Open-Source Operating System for Wireless Embedded Sensor Networks. `www.tinyos.net`.

[190] Gilman Tolle, Joseph Polastre, Robert Szewczyk, David Culler, Neil Turner, Kevin Tu, Stephen Burgess, Todd Dawson, Phil Buonadonna, David Gay, and Wei Hong. A Macroscope in the Redwoods. In *Proceedings of the 3rd International Conference On Embedded Networked Sensor Systems (SenSys 2005)*, pages 51–63, San Diego, CA, USA, November 2005.

[191] Vlad Trifa, Alexander N. G. Kirschel, Charles E. Taylor, and Edgar E. Vallejo. Automated Species Recognition of Antbirds in a Mexican Rainforest using Hidden Markov Models. *Journal of the Acoustical Society of America*, 123(4):2424–2431, April 2008.

[192] Simon Tschirner, Liang Xuedong, and Wang Yi. Model-Based Validation of QoS Properties of Biomedical Sensor Networks. In *Proceedings of the 8th ACM International Conference On Embedded Software*, pages 69–78, Atlanta, GA, USA, October 2008.

[193] M. L. Tsetlin. *Finite Automata and Modeling the Simplest Forms of Behavior*. PhD thesis, St. Petersburg Department of V.A. Steklov Mathematical Institute, Russian Academy of Sciences, 1964.

[194] Daniela Tulone and Samuel Madden. PAQ: Time Series Forecasting for Approximate Query Answering in Sensor Networks. In *Third European Workshop on Wireless Sensor Networks (EWSN 2006)*, pages 21–37, Zurich, Switzerland, February 2006.

[195] Michael Unser. Sampling - 50 Years After Shannon. *Proceedings of the IEEE*, 88(4):569–587, April 2000.

[196] Tijs Van Dam and Koen Langendoen. An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensing Systems (SenSys 2003)*, pages 171–180, Los Angeles, CA, USA, November 2003.

[197] Kristof Van Laerhoven, Albrecht Schmidt, and Hans-Werner Gellersen. Multi-Sensor Context Aware Clothing. In *Proceedings of the 6th International Symposium on Wearable Computers (ISWC 2002)*, pages 49–56, Linz, Austria, September 2002.

[198] Mehmet C. Vuran, Özgür B. Akan, and Ian F. Akyildiz. Spatio-Temporal Correlation: Theory and Applications for Wireless Sensor Networks. *Computer Networks*, 45(3):245–259, June 2004.

[199] Masaaki Wada, Katsumori Hatanaka, and Masashi Toda. Developing a Water Temperature Observation Network based on a Ubiquitous Buoy System to Support Aquacultures. *Journal of Communications*, 3(5):2–11, October 2008.

[200] Peng-Jun Wan and Chih-Wei Yi. Coverage by Randomly Deployed Wireless Sensor Networks. *IEEE/ACM Transactions on Networking (TON), Special Issue on Networking and Information Theory*, 14:2658–2669, June 2006.

[201] Geoffrey Werner-Allen, Jeff Johnson, Mario Ruiz, Jonathan Lees, and Matt Welsh. Monitoring Volcanic Eruptions with a Wireless Sensor Network. In *Proceedings of the 2nd European Workshop on Wireless Sensor Networks (EWSN 2005)*, pages 108–120, Instanbul, Turkey, January 2005.

[202] Kamin Whitehouse, Gilman Tolle, Jay Taneja, Cory Sharp, Sukun Kim, Jaein Jeong, Jonathan Hui, Prabal Dutta, and David Culler. Marionette: Using RPC for Interactive Development and Debugging of Wireless Embedded Networks. In *Processing of the 5th International Conference on Information Processing in Sensor Networks (IPSN/SPOTS 2006)*, pages 416–423, Nashville, TN, USA, April 2006.

[203] Rebecca Willett, AlineMartin Martin, and Robert Nowak. Backcasting: Adaptive Sampling for Sensor Networks. In *Proceedings of the Third International Symposium on Information Processing in Sensor Networks (IPSN 2004)*, pages 124–133, Houston, TX, USA, April 2004.

[204] Rebecca M. Willett and Robert D. Nowak. Platelets: a Multiscale Approach for Recovering Edges and Surfaces in Photon-Limited Medical Images. *IEEE Transactions on Medical Imaging*, 22(3):332–350, March 2003.

[205] Gouliang Xing, Xiaorui Wang, Yuanfang Zhang, Chenyang Lu, Robert Pless, and Christopher Gill. Integrated Coverage and Connectivity Configuration for Energy Conservation in Sensor Networks. *ACM Transactions on Sensor Networks (TOSN)*, 1(1):36–72, August 2005.

[206] Zixiang Xiong, Angelos D. Liveris, and Samuel Cheng. Distributed Source Coding for Sensor Networks. *IEEE Signal Processing Magazine*, 21(5):80–94, September 2004.

[207] Yingqi Xu, Julian Winter, and Wang-Chien Lee. Dual Prediction-Based Reporting for Object Tracking Sensor Networks. In *Pro-*

ceedings of the 1st International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous 2004), pages 154–163, Boston, MA, USA, August 2004.

[208] Yong Yao and Johannes Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. *SIGMOD Record*, 31(3):9–18, September 2002.

[209] Fan Ye, Gary Zhong, Jesse Cheng, Songwu Lu, and Lixia Zhang. PEAS: A Robust Energy Conserving Protocol for Long-Lived Sensor Networks. In *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS 2003)*, pages 28–37, Providence, RI, USA, May 2003.

[210] Honghai Zhang and Jennifer C. Hou. On deriving the upper bound of α-lifetime for large sensor networks. In *Proceedings of the 5th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2004)*, pages 121–132, Tokyo, Japan, May 2004.

[211] Feng Zhao and Leonidas Guibas. *Wireless Sensor Networks: An Information Processing Approach*. Morgan Kaufmann, San Francisco, CA, USA, 2004.

[212] Feng Zhao, Jaewon Shin, and James Reich. Information-Driven Dynamic Sensor Collaboration for Tracking Applications. *IEEE Signal Processing Magazine*, 19(2):61–72, March 2002.

[213] Qing Zhao and Lang Tong. Energy-Efficient Information Retrieval for Correlated Source Reconstruction in Sensor Networks. *IEEE Transactions on Wireless Communications*, 6(1):157–165, January 2007.

Curriculum Vitae

Silvia Santini

Personal Data

Date of Birth	November 15, 1978
Birthplace	Rome, Italy
Citizenship	Italian

Education

2004-2009	PhD Student at the Department of Computer Science of the *ETH Zurich*, Switzerland
May 29, 2004	Master of Science (Laurea) in Telecommunications Engineering (graduation mark: 110/110 cum laude)
2001-2004	Studies of Telecommunications Engineering at the *Sapienza University of Rome*, Italy
2000-2001	Exchange Student at the Department of Electrical Engineering of the *ETH Zurich*, Switzerland
1997-2000	Studies of Telecommunications Engineering at the *Sapienza University of Rome*, Italy
July 1997	High School Diploma (Maturità Scientifica) with Advanced Studies in Mathematics, Computer Science and Physics (graduation mark: 60/60)
1992-1997	*Liceo Scientifico Statale Tullio Levi Civita*, High School in Rome, Italy
1989-1992	*Scuola Media Statale Gioacchino Rossini*, Secondary School in Rome, Italy
1984-1989	*Scuola Elementare Statale Vittorino Chizzolini*, Elementary School in Rome, Italy

Employment

2004-2009	Teaching and Research Assistant at the Department of Computer Science of the *ETH Zurich*, Switzerland
2003	Junior Researcher in the Imaging Systems group at *Philips Research Center*, Aachen, Germany
2002-2004	Undergraduate Assistant at the Department of Engineering of *Sapienza University of Rome*, Italy
2001	Undergraduate Assistant at the Department of Mechanical and Process Engineering of the *ETH Zurich*, Switzerland