Collaborative Wireless Sensor Networks in Industrial and Business Processes

Mihai Marin-Perianu

Composition of the Graduation Committee:

Prof. Dr. Ir.	G.J.M.	Smit	(UT, CAES)
Dr.	P.J.M.	Havinga	(UT, PS)
Prof. Dr. Ir.	Th.	Krol	(UT, CAES)
Prof. Dr.	J.	van Hillegersberg	(UT, IS&CM)
Prof. Dr.	J.J.	Lukkien	(TU Eindhoven)
Prof. Dr. Ing.	G.	Tröster	(ETH Zürich, Switzerland)
Prof. Dr. Ing.	N.	Ţăpuş	(UPB, Romania)
Prof. Dr. Ing.	M.	Beigl	(TU Braunschweig, Germany)

This work was conducted within the EU projects CoBIs (IST 004270), AWARE (IST 2006-33579), e-SENSE (contract no. 027227) and SENSEI (contract no. 215923).

Keywords: Wireless Sensor Networks, Industrial and Business Processes, Reliability, Activity Recognition, Fuzzy Logic.

Cover Design: Newblack, `www.newblack.ro`.

Printed by Wöhrmann Print Service.

ISBN 978-90-365-2746-0

CTIT PhD Thesis Series Number 08-131

COLLABORATIVE WIRELESS SENSOR NETWORKS IN INDUSTRIAL AND BUSINESS PROCESSES

DISSERTATION

to obtain
the degree of doctor at the University of Twente,
on the authority of the rector magnificus,
prof.dr. W.H.M. Zijm,
on account of the decision of the graduation committee,
to be publicly defended
on Thursday the 6th of November 2008 at 16.45

by

Mihai Marin-Perianu

born on the 6th of May 1979

in Bucharest, Romania

Dit proefschrift is goedgekeurd door

Prof. Dr. Ir.	G.J.M.	Smit	(promotor)
Dr.	P.J.M.	Havinga	(assistent-promotor)

Abstract

Wireless Sensor Networks (WSNs) create the technological basis for building pervasive, large-scale distributed systems, which can *sense* their environment in great detail, *communicate* the relevant information via the wireless medium, *reason* collectively upon the observed situation and *react* according to the application-specific goals. Embedding sensing, processing and communication in one tiny device (the *sensor node* or simply *mote*), which can subsequently collaborate with peers and build a self-organizing, self-healing network, stimulates a long list of applications from various domains, ranging from environmental monitoring to industrial processes, and even further to cognitive robotic systems or space exploration.

At first glance the complexity of such applications is overwhelming, given the serious resource limitations of sensor nodes, in terms of computational power, storage space, radio performance and battery power. However, WSNs have a unique feature that balances the inherent resource limitations: the ability of in-network *collaboration* at scale. Through collaboration WSNs can organize efficiently, prolong system lifetime, handle dynamics, detect and correct errors, all with the final goal of eventually executing reliably the user application.

Following this line, researchers devised an impressive number of collaborative WSN algorithms and protocols in recent years. Significant progress has also been made on the market side, so that nowadays we can claim that WSNs are no longer just lab prototypes. Standardization initiatives (such as IEEE 802.15.4) are being put into practice and the general industry trend strongly suggests that the epoch of pioneering research in building and experimenting with "motes" is approaching an end. It is now the logical time for *system integration* and for *creating bridges to connected fields*.

This thesis focuses on WSN integration in industrial and business processes, and, more specifically, on exploring collaborative techniques to make WSNs more reliable, intelligent, effective and easy-to-use in industry-related scenarios.

In short, the contributions of this thesis are:

- **Market analysis** of several industrial fields of interest for WSNs: enterprise systems, transport and logistics, industrial automation, safety-critical processes, automotive industry and automatic meter reading.
- **Service-oriented architecture** for integrating different WSN platforms and exposing their functionality in a uniform way to the back-end system. In the European project CoBIs, we demonstrated the complete system, including two WSN platforms, business rules support, reconfiguration via reliable multicast data dissemination, uniform gateway translation using UPnP and final integration into enterprise software.
- **Reliable data and code dissemination** – a multicast protocol for disseminating data reliably to groups of nodes with minimal energy expenditure. Our protocol addresses both end-to-end and local error control, and applies the multicast communication model for increased scalability and flexibility.
- **Rule-based inference** for distributed situation assessment and event detection at the point of action. This contribution brings two innovations to the field: business rule support and distributed fuzzy logic inference engine on wireless sensor nodes.
- **Distributed activity recognition** using fuzzy-enabled WSNs that become aware of the user actions and provide context-aware assistance. In this work we apply fuzzy inference to the unreliable sensor data and incorporate temporal order knowledge about the sequences of operations, in order to increase the overall accuracy of the recognition system.
- **Mobile team coordination** – a low-cost, low-power method for movement coordination based on inertial sensing, wireless communication and fuzzy control. To our knowledge, this is the first solution considering solely inertial sensors and running entirely on resource-constrained sensor nodes.

Samenvatting

Draadloze sensornetwerken (Wireless Sensor Networks, WSNs) vormen de technologische basis voor grootschalig gedistribueerde systemen die hun omgeving heel gedetailleerd kunnen *waarnemen*, de relevante informatie via een draadloos medium kunnen *communiceren*, gezamenlijk over de geobserveerde situatie kunnen *redeneren* en applicatiegericht kunnen *reageren*. Het samenvoegen van waarneming, verwerking en communicatie van gegevens in één minuscuul apparaatje (de *sensor node* of eenvoudigweg *mote*) dat met andere nodes kan samenwerken om zo een zelforganiserend en zelfhelend netwerk op te zetten opent mogelijkheden in diverse toepassingsgebieden. Daarbij valt te denken aan toepassingen variërend van milieubewaking tot industriële processen en zelfs verkenningen in de ruimte of cognitieve robotische systemen.

Op het eerste gezicht is de complexiteit van dergelijke applicaties overweldigend, gegeven de zeer beperkte middelen, wat betreft rekenkracht, opslagcapaciteit, radiobereik, radiobandbreedte en batterijvermogen, die sensor nodes ter beschikking hebben. Draadloze sensornetwerken hebben echter een unieke eigenschap die de inherente beperkte mogelijkheden van individuele nodes compenseert: het vermogen om *samen te werken* in een grootschalig netwerk. Door middel van samenwerking kunnen draadloze sensornetwerken zichzelf efficiënt organiseren, hun levensduur verlengen, omgaan met dynamisch gedrag van de omgeving en fouten corrigeren. Hierop voortbouwend hebben onderzoekers in de laatste jaren een indrukwekkend aantal algoritmes en protocollen ontwikkeld die de samenwerking in draadloze sensornetwerken kunnen realiseren. Er is ook significante vooruitgang geboekt met het op de markt brengen van deze technologie.Initiatieven voor standaardisatie (zoals IEEE 802.15.4) worden in de praktijk gebracht en de algemene trend in de industrie duidt er sterk op dat het tijdperk van baanbrekend onderzoek in het bouwen en experimenteren met "motes" zijn einde nadert. De tijd is rijp voor *systeemintegratie* en voor het

slaan van bruggen naar gerelateerde vakgebieden.

Dit proefschrift richt zich op het integreren van draadloze netwerken in industriële en zakelijke processen. We besteden vooral aandacht aan het verkennen van nieuwe algoritmes voor het laten samenwerken van sensor nodes. Dit heeft als doel draadloze sensornetwerken in industriële situaties betrouwbaarder, intelligenter, effectiever en gebruiksvriendelijker te maken. Samenvattend zijn de bijdragen van dit proefschrift als volgt:

- Een **marktanalyse** van diverse industriële interessegebieden voor WSNs: bedrijfssystemen, transport en logistiek, industriële automatisering, veiligheidskritieke systemen, de auto-industrie en het automatisch uitlezen van verbruiksmeters.
- Een **dienstgerichte architectuur** voor het integreren van verschillende WSN platformen. Bovendien kan deze architectuur de functionaliteit van zulke platformen op een uniforme wijze aan het achterliggende systeem presenteren.
- Een zeer energiezuinig multicast protocol voor **betrouwbare gegevens- en programmacodeverspreiding** over groepen sensor nodes. Ons protocol implementeert zowel foutafhandeling voor verbindingen over het hele netwerk alsook voor verbindingen tussen de nodes. Tevens past dit protocol het multicast communicatiemodel toe voor betere schaalbaarheid en flexibiliteit.
- Een **op regels gebaseerde deductietechniek** voor situatiebeoordeling en voor het direct detecteren van gebeurtenissen. Deze bijdrage behelst een tweetal innovaties: ondersteuning voor business rules en een gedistribueerde fuzzy logic implementatie voor draadloze sensornetwerken.
- **Gedistribueerde activiteitsherkenning** met WSNs voorzien van fuzzy logic. Hiermee worden WSNs zich bewust van de activiteiten van de gebruiker en kunnen ze de gebruiker assisteren op een wijze die overeenstemt met de huidige context. In dit werk passen we fuzzy logic toe op de onbetrouwbare sensor gegevens. Tevens gebruiken wij kennis over de volgorde waarin handelingen binnen een activiteit uitgevoerd worden om de algehele nauwkeurigheid van het herkenningssysteem te verbeteren.
- **Mobiele team coördinatie** - een goedkope en energiezuinige methode voor bewegingscoördinatie gebaseerd op bewegingsmetingen, draadloze communicatie en fuzzy logic besturing. Voor zover wij weten is dit de eerste oplossing die enkel bewegingssensoren gebruikt en volledig draait op sensor nodes die sterk beperkt zijn in hun mogelijkheden.

Acknowledgments

All that get here - to write the acknowledgments of their PhD thesis - must recognize that this is no simple task. After all, let's be honest, everyone reads the acknowledgments section, so few delve into the next 200 pages! If only I think of how many people I should thank for supporting me throughout the last four years, I already have a problem with the thesaurus of the verb "to thank" - English is sometimes so poor in synonyms!

But before I start listing names and memories, I should first explain what and how my PhD was. Well, as any other thing in this life, it was a combination of great and not-so-great experiences. But what matters is the overall feeling, and this is extremely positive. So, to the ever-recurring question *"Was it a good choice to do your PhD?"*, I will always answer a definite *YES*! During the last four years, I felt that I am learning new things and improving myself everyday. The international environment was a fantastic experience. Above all, I felt that I was not doing a "from 9 to 5 job" (which I think will never be my favorite choice), but I was benefiting from my work, I was continuously extending knowledge even further. Now, I also mentioned not-so-great experiences, and every PhD candidate should be aware of them. There are times of confusion, when you discover that what you thought to be a revolutionary idea is actually literature. Also, the way is paved with disappointments: it is hard to find people enthusiast about your results, reviewers do not find any useful contribution in your half-year hard work, simulations and experiments drag and seem to never end, etc. These bring many people into the "valley of despair" around the second year of their PhD, and then many of them quit... Therefore, for helping me to arrive successfully at the end of this strenuous path, I have to thank to a long list of people.

First of all, to my promotor, Gerard Smit, who gave his consent for submitting this thesis. I was impressed with the detail of his comments, very precise and to the point, although he had to read a large part of my thesis during a

long flight to Korea (not a very entertaining occupation, I guess). Moreover, I have to admire his open-mindedness for allowing me to "try my luck" with the business-oriented chapter, even though this is not core research in our group.

To Paul Havinga, my assistent-promoter and supervisor, I would have more thanks to express than can fit in this section. From the first five minutes of my interview, long time ago, I knew that we are passionate about similar things: imagine new solutions that are usable in real life, design, study, refine and implement them. In other words, produce tangible results from science. Paul holds all the necessary qualities for being a perfect supervisor, mentor, friend and interlocutor. I guess this is why so many people want to talk to him all the time, so that he became the person with the busiest agenda I have ever seen! Despite this, he always finds some time to discuss my "urgent" problems, give support, out-of-the-box ideas and, last but not least, spread a contagious optimism. Paul, I am grateful for all these and I hope we will continue our collaboration for many years... and, still, I hope to get you to visit Romania some time!

I would also like to express my acknowledgments to all the members of my graduation committee. Thijs Krol was my initial promotor and offered me a warm welcome in the CAES group at the beginning of my PhD. Gerhard Tröster accommodated both Raluca and I in the Wearable Computing Lab at ETH Zürich. We are grateful to him for a great work experience in an excellent research environment, but also for an unforgettable group hike in the Swiss Alps! Nicolae Ţăpuş taught some of the best courses in our university years and was always interested in how our research was progressing. With Michael Beigl, I collaborated in our favorite CoBIs project and, later on, we met at many interesting conferences all over the globe. With Johan Lukkien, I had useful discussions about software-related aspects of WSNs. Jos van Hillegersberg helped me with his expertise on business information systems to better shape the market analysis chapter.

During my PhD, I was fortunate to be involved in no less than four European projects: CoBIs, AWARE, e-SENSE and SENSEI. I met and collaborated with so many people, that I do not have the necessary space to thank each of them: Luciana Moreira Sá de Souza, Patrik Spieß, Stephan Haller, Jens Müller, Dominique Guinard (SAP), Till Riedel, Christian Decker (University of Karlsruhe), Guido Stromberg (Infineon Technologies), Clemens Lombriser, Oliver Amft, Andreas Bulling, Martin Kusserow, Daniel Roggen, Bert Arnrich (ETH Zürich), Stuart Kininmonth (AIMS), Tjerk Hofmeijer, Leon Kleiboer, Mark Bijl, Wouter van Kleunen (Ambient Systems).

What really compensated for the hopeless rainy weather throughout these

four years was the international “gang” that I met here, in Twente. It has been an amazing experience, both at the office and in outings, parties, sports, trips together etc. My Romanian friends - Ileana, Ştefan, Oana, Vali, Georgiana, Andreea, Eugen - made me forget about the distance to my home country, made our parties so lively, and supported Raluca and I in every possible way.

Despite our historical fights (always heroically won by Romanians, of course!), we got along very well with an ever-growing number of Turkish people: Özlem, Mustafa, Sinan, Ayşegül, Kamil, Ayşe, Seçkin, Cem - “Teşekkürler” to all of you! Michel was our friendly Dutch “leraar”, always enthusiast, even though our progress was so pathetic. I should not forget Lodewijk and Tjerk, who also gave us some entertaining Dutch lessons. Our Indian friends always bring color and spice to our parties: Kavitha, Kiran and the “stick-couple”, which became a “stick-trio” - Anindita, Samhita and Supriyo. Nirvana and Maria are great friends and lively colleagues, and perhaps the fastest writers of project proposals in our group. Stephan shares my passion for “toys”, from inertial sensors to Ferrari cars, and I foresee a lot of fun applications that we will develop together. With Yang, I enjoy talking about China and playing badminton, and I am sure that soon he will play much better than me. I always had interesting discussions with Hans and Pierre, whether they were about Himalayas, distributed architectures, the Mexican jungle or real time systems. Of course, so many bureaucratic things would have overwhelmed me without the precious help of our dear secretaries: Nicole, Thelma and Marlous.

I would also like to acknowledge a number of good friends and former colleagues scattered around the world: Sinan, Anka, Ari, Ha, Roland, Vasughi, Tim, Law. Last but not least, our old friends from Romania - “gaşca” - that never forgot us: Ioana, Radu, Dora, Claudia, Doru, Mădă, Cip, Nicoleta, Adiţă.

The support of my family was essential. This thesis is first of all dedicated to my parents, who taught me so much and encouraged me all the way thorough the difficult Math and Physics Olympic contests in high school. My father still helps us with complicated hardware design in our projects, simply because he is the best Engineer I have ever seen. My dear sister, Irina, is always by our side and, together with Andrei, worked out so many brilliant logos, websites and any graphical stuff we needed - they are both really talented designers! My grandparents are always in my memory, although some of them passed away, I will never forget their love. I would also like to thank my numerous relatives from Ştefăneşti and Brăila for receiving us with great enthusiasm every Easter and Christmas. At last, to my parents in law, Dana and Puiu, a big thanks for the unforgettable trips we made together - let’s keep it going in the future!

Now, I should end up this section, otherwise everyone will become bored.

Hoping that I did not forget anyone, I can write the last phrase:

To Raluca, my love, to sail together in all our journeys, forever!

Contents

Chapter 1

Introduction

Personal computers, mobile telephony and the Internet made the vision of *everyone* being networked real, at such a level of quality and speed that people could only dream of, one hundred years ago. Nowadays, another dream starts to become reality: the dream of networking *everything*. The limits of this *everything* are set only by our imagination: containers [129], trucks [79], robots [45], coffee cups [44], zebras [118], corals [49], firefighters [13], energy meters [6], etc. Wireless Sensor Networks (WSNs) [76, 100, 35] and complementary pervasive technologies [1] created the technological basis for all these applications by embedding sensing, processing and communication in one tiny device (the *sensor node*) and by executing tasks in a distributed, collaborative manner within the network.

To start with, a sensor node typically contains a microcontroller and a radio transceiver (sometimes combined in a single chip package), digital and analog interfaces for connecting sensors and actuators, batteries and, optionally, additional storage memory and communication ports (e.g. USB, RS-232). As a standalone device, a sensor node has very limited resources in terms of:

- Computational power (typically 8 or 16 bit CPUs at 4-8 MHz)
- Storage space (in the order of 10kB RAM and 48kB FLASH).
- Radio data rates (50-100 kbps) and coverage (typically 20-200 m).

[1] Throughout this work, we will denote by Wireless Sensor Networks and the WSN acronym the entire spectrum of related technologies, such as Wireless Sensor and Actuator Networks [34], Smart Collaborating Objects [154], Internet of Things [16], etc.

- Available battery power (approximately 1500 mAh with typical batteries).

These limitations pose serious challenges in developing complex applications on sensor nodes. To complicate the situation even further, typical usage scenarios involve deployments in harsh conditions, such as extreme natural environments, military areas, industrial sites, thus increasing the likeliness of failures. By failures we should understand not only the complete damage of a node, but also various errors that adversely affect the system performance, such as the deterioration of the wireless channel, inaccurate or even faulty sensor readings, insufficient battery power for performing certain tasks, etc. All these problems turn an individual sensor node into a very undependable element from the application developer perspective. The one and only advantage is the possibility of combining many such undependable nodes into a distributed system, which accepts dynamics, self-organization and self-healing as basic mechanisms to eventually execute reliably the user application.

Networking and cooperation are therefore essential to achieve the desired functionality in WSNs. Researchers and developers devised an impressive number of distributed algorithms and protocols that optimize in various ways the resource utilization of sensor nodes, with the final goal of prolonging the system (or network) lifetime while still fulfilling the application requirements. Significant progress has been made in recent years in this regard and WSNs are no longer just simulations or lab prototypes. However, the future of the technology in the market is yet undecided. The situation can be briefly summarized as follows:

- The good news: WSN companies and research organizations report successful installations, the operating systems and network stacks are functional (with certain scalability limits), standardization efforts materialize (with IEEE 802.15.4 [15] and ZigBee [31] growing popular), the number of spinoffs continues to increase steadily.
- The bad news: sensor nodes are far from costing just a few cents, dropping them from the airplane is not a popular deployment method, installation and packaging can add unexpectedly high costs, the connection to the back-end applications lacks standard interfaces and results into tedious, platform-specific adaptation of software.

Despite these problems, the market awareness and the range of potential applications extended significantly since the dawns of WSN technology. Even

though no obvious "killer application" exists at the moment, WSNs can still be *the next big thing* [90].

1.1 WSNs in Industrial and Business Processes

Initially perceived as an effective tool for monitoring large geographical areas in detail, WSNs have recently attracted the interest of industrial and business users. The European project CoBIs (Collaborative Business Items) [4] investigated the potential usage of WSNs in industrial and business application domains. An important result was that *service-oriented WSNs* represent a powerful paradigm that can determine the successful uptake of this technology into the market. The service abstraction in the context of WSNs has two important advantages:

- Services are easy to understand and use, as they expose only the relevant functionality an hide the unnecessary low level details.
- A service-oriented architecture gives a better view of which functional components are already available and which still need to be developed and, additionally, facilitates the uniform interfacing between such components.

Based on the CoBIs architecture [4], Figure 1.1 groups the most important WSN services by their commonalities in supporting the applications. We distinguish the following building blocks:

- The *core*, including the basic hardware and low level software present on a typical node.
- The *basic services*, which render elementary functionality, mainly related to networking. Together, the core and the basic services form the sensor node platform.
- The *complex services*, which provide a rich set of functions to the application. The complex services rely heavily on collaboration among nodes. In contrast to the core and the basic services, they can be devised as platform-independent components.
- The *service execution support* confers flexibility in programming the WSN and can handle heterogeneous platforms.

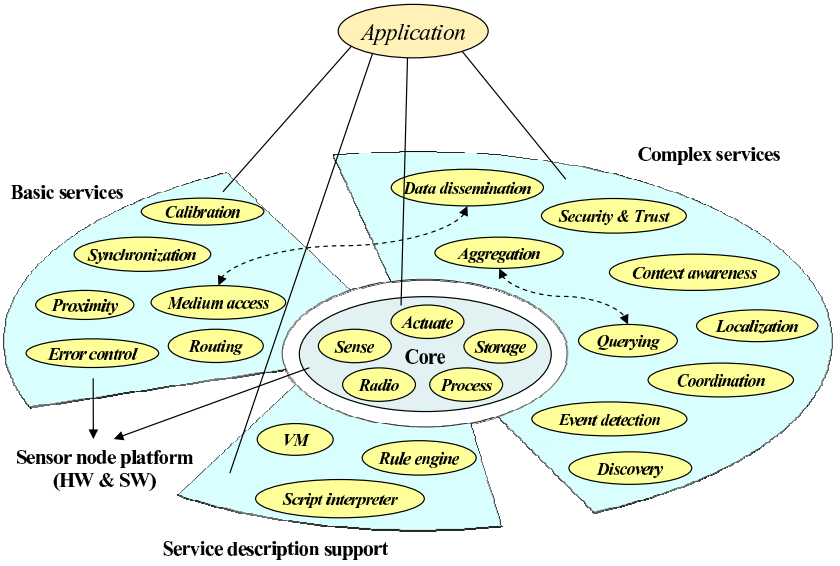


Figure 1.1: WSN functional components classified into four building blocks: core, basic services, complex services and service execution support. The components are often interacting (for example querying and aggregation) in order to provide an efficient solution.

This classification does not exhaustively accommodate all imaginable services. It relies on the idea that basic services are compact, often platform-dependent code modules that render elementary functionality(e.g. MAC), while complex services represent truly collaborative software that implements higher level logic and could be specified in a platform-independent way (e.g. Querying).

While basic services are already functional and embedded in available sensor node platforms, complex services and service execution support are still the object of intensive research. This thesis advances the state of the art by proposing novel solutions for several topics, namely data dissemination, event detection, context-awareness, coordination and rule-based inference engines. In addition, we define a generic architecture for simplifying the integration of WSNs and the added functionality with the existing back-end systems.

1.2 Research Issues

The main research question that this thesis addresses is:

Question 1. *How can WSNs be assimilated in complex industrial and business processes, so that they bring a clear technological advance to the user, together with commercial added value?*

This research problem articulates along multiple axis of interest and can be divided into several sub-questions:

Question 2. *What is the worldwide industrial and business market perspective with respect to WSN technology? What are the main trends and opportunities and, above all, which are the key requirements and challenges for having WSNs implemented in such real-life applications?*

Question 3. *Starting from the identified requirements, what is the right architecture for integrating heterogeneous WSNs with the existing back-end systems and thus enabling what we call "the real-time enterprise"?*

Question 4. *How can we improve the overall reliability of WSNs, given the inherent limitations of sensor nodes, the unpredictability of the wireless communication, the sensor inaccuracy and the harsh industrial environments?*

Question 5. *How can WSNs exploit collaboration and local interaction so that they make the term "intelligent sensors and actuators" real to the industrial automation community? Building on context-awareness and sensor-actuator coordination, how can WSNs simplify the tasks of the user, improve the quality of the given process and provide novel functionality?*

1.3 Contributions

This thesis takes a top-down approach in investigating and answering Question 1 and subsequently Questions 2 to 5:

Contribution 1 (The big picture) – *Chapter 2.* As a first step, we conduct a market study in several major industrial and business-related fields: enterprise systems, transport and logistics, safety-critical systems, industrial automation, automotive industry and automatic meter reading systems. In each field we overview the current market status and estimated trends, we identify potential applications of WSNs, and eventually we derive the key requirements

that WSNs have to meet in order to get a significant uptake. As a whole, Contribution 1 gives a detailed, even though not exhaustive, answer to Question 2.

Contribution 2 (Architecture) – *Chapter 3.* Based on the successful experiences with the EU-funded project CoBIs, we present a three-layer service-oriented architecture (SOA) that facilitates the integration of different WSN platforms in industrial and business processes. The functionality of the WSNs is exposed to the back-end system in a uniform way, by using the Universal Plug and Play (UPnP) standard. The ultimate goal is to delegate well-defined parts of the business logic to the low-cost embedded devices, and thus reduce the process execution costs and improve the response time in safety-critical situations. We present both practical tests and application trials that confirm the feasibility of our solution and the benefits of multi-platform integration. Therefore, Contribution 2 corresponds to Question 3.

Starting from the proposed SOA, we design, develop and evaluate four functional components within the WSN layer, summarized below as Contributions 3 to 6. The two keywords that best characterize these components are *reliability* and *distributed intelligence*. Consequently, Contributions 3 and 4 relate to Question 4, whereas Contributions 5 and 6 relate to Question 5.

Contribution 3 (Reliable data and code dissemination) – *Chapter 4.* Industrial applications have stringent requirements in terms of communication reliability. In addition, there is often the need to reconfigure the WSN from the back-end systems, by altering the tasks on the nodes or just updating some parameters. Therefore, Contribution 3 addresses a critical component in WSNs: reliable data and code dissemination. Starting from practical experiments with reliable data delivery in WSNs, we design, evaluate and implement a reliable multicast dissemination solution. Furthermore, we analyze the performance of the entire network stack (data link, topology control and reliable dissemination) and study the impact of the deployment properties, as well as of the irregularities of the real wireless environment.

Contribution 4 (Rule-based inference) – *Chapter 5.* Sensor nodes have the ability of executing an inference engine, through which they can assess the observed situation and even make decisions at the point of actions. This is an important feature, as it makes real the term "intelligent sensor" to the industrial community and leverages the load on the back-end system. Based on the widely accepted business rule paradigm, we propose a lightweight business rule inference engine for WSNs, which offers the user the possibility to express easily the application logic and, moreover, to change it with minimal overhead.

In a second step, we extend the flexibility and robustness of rule-based inference by utilizing fuzzy logic. The outcome is a distributed fuzzy logic inference engine that fuses sensor data and neighborhood observations to produce reliable results for the generic problem of event detection.

Contribution 5 (Activity recognition) – *Chapter 6.* Context-aware WSNs contribute to augmenting the human-machine interaction and lay down the foundations of future sophisticated and adaptive cognitive systems. In particular, Contribution 5 focuses on the concrete industrial application of assembling and testing car body parts at a car manufacturing site. We design and evaluate a distributed activity recognition system that incorporates wireless sensor nodes worn by the workers, embedded into the tools and deployed within the infrastructure. By recognizing the user activities in real time, the system can provide prompt assistance to workers, supervise safety-critical process steps and accelerate the learning curve of new personnel. The sensor nodes use fuzzy logic for reliably detecting and classifying the user activities online. The system outperforms non-fuzzy methods considered in previous work, especially when including temporal order knowledge about the sequences of user operations into the inference process.

Contribution 6 (Team coordination) – *Chapter 7.* Cooperation and coordination are keywords when imagining various scenarios for wireless sensor and actuator networks, ranging from down-to-earth cooperative surveillance to science-fiction planet exploration. However, the two notions are covered in little extent by the research literature. To make a step further, Contribution 6 addresses the problem of distributed movement coordination of vehicles equipped with wireless sensor nodes. The final goal is to have a self-organizing team (or swarm) of nodes that maintain a formation by periodically exchanging their sensed movement information. Each node features low-power inertial sensors, from which it computes speed and orientation online. The leader vehicle periodically transmits these measures to the followers, which implement a lightweight fuzzy logic controller for imitating the leader's movement pattern. The solution is not restricted to vehicles on wheels, but supports any moving entities capable of determining their velocity and heading, thus opening promising perspectives for machine-to-machine and human-to-machine spontaneous interactions in the field. We report in detail on all development phases, covering design, simulation, controller tuning, inertial sensor evaluation, calibration, scheduling, fixed-point computation, debugging, benchmarking, field experiments and lessons learned.

1.4 Methodology

The methodology of work adheres to the same top-down approach. By starting with the generic architecture definition, we can decompose the global problem into several subproblems and study these separately. For each subproblem, we adopt the following methodological steps:

1. Define the primary and secondary objectives.
2. Survey the related work and identify the advantages, drawbacks and missing gaps.
3. Prototype the initial ideas and assess practically their feasibility.
4. Evaluate theoretically and/or via simulations the performance factors.
5. Implement and study the system on a real sensor node platform.
6. Draw the conclusions and discuss future research directions.

Throughout our work, we found steps 3 and 6 particularly important. Although the practical nature of WSN research is widely recognized, many studies unfortunately still lack real life evaluation. Prototyping the initial ideas can provide valuable insight into the real problems and trigger design modifications in the early phase of development. Likewise, implementing the proposed system, even at a small scale, gives a much more realistic view of the actual performance and possible issues than simulation studies. It it therefore our strong argument that the results of practical work are worth the effort of delving into WSN programming.

Chapter 2

Market Analysis

This chapter presents a market overview of several major industrial and business-related fields: enterprise systems, transport and logistics, safety-critical systems, industrial automation, automotive industry and automatic meter reading systems. In each field we present the current market trends, we identify potential applications of WSNs, and eventually we derive the key requirements that WSNs have to meet in order to get a significant uptake.

2.1 Introduction

WSNs raised a wave of enthusiasm throughout the research community. This alone is not enough, however, to measure the real potential of this technology push. The fundamental question from the business analyst point of view is: *Will WSNs be a disruptive technology and impact on a wide spectrum of world's markets, or it will remain a research prototype platform?* Currently, there is no "yes-or-no" answer to this question and predictions are prone to be speculative, as the market position versus WSNs is undecided.

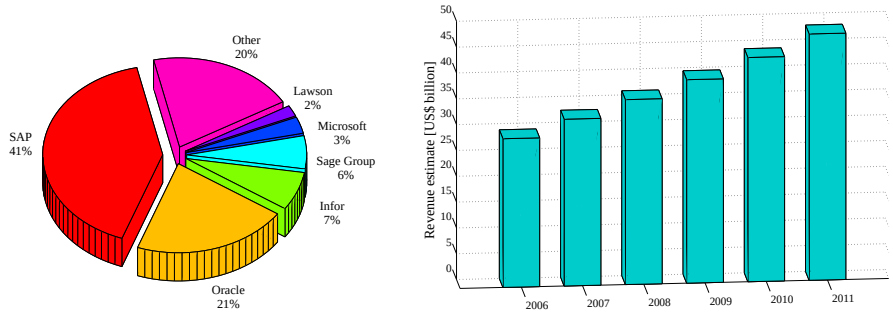
In order to have a better understanding of the puzzle, we overview in this chapter six application domains with large potential for WSN technology: enterprise systems, transport and logistics, industrial automation, safety-critical processes, automotive industry and automatic meter reading. We select these application domains because they account for clear initiatives of introducing the WSN technology, ranging from small pilot projects to concrete standardization efforts. This study, therefore, gives a discrete image of the WSN potential in industrial and business-related areas, and does not pretend to build up an exhaustive survey of the worldwide market situation.

For each of the aforementioned application domains, we present: (1) a brief description and market trends, (2) the perspectives of WSNs in the field and (3) the most important requirements and practical challenges. Based on this analysis, we formulate in the end what would be, in our opinion, the most probable evolution of WSN arena on the short, medium and long term.

2.2 Enterprise Systems

Definition, facts and figures. Enterprise Resource Planning (ERP) systems are described as "one of the most pervasive, extensive and complex organizational information systems (IS) nowadays" [165]. Broadly speaking, ERP systems address the problem of fragmentation of information in large business organizations [68] and therefore their ultimate goal is to integrate all data and processes into a unified system. The major application segments of an ERP system include: enterprise management, supply chain management, product lifecycle management, sourcing and procurement, customer management and human resources.

The ERP market amounted to over US$ 28 billion in 2006, with total revenue growing by 14%, and is estimated to exceed US$ 47 billion by 2011 (see Figure 2.1). This substantial growth is driven by two factors: (1) the continued



(a) (b)

Figure 2.1: (a) Top ERP vendors in 2006; (b) Estimate of ERP market evolution (source AMR Research [97]).

investment among large corporations due to globalization and centralization, and (2) the increasing participation of small and midsize business (SMB) segment in the global market [97].

WSN technology perspectives. Enterprise systems already make extensive use of RFID technology for item identification and tracking [50]. Major ERP vendors, such as SAP, investigate the potential usage of WSN technology, through which an even larger range of industrial and business processes can benefit from relocating logic to the point of action [4, 59]. Delegating parts of the business functionality to distributed, low-cost devices has several important benefits, such as: (1) reducing the load on the back-end system, (2) decreasing the process execution and transactional costs, and (3) providing better response in time-critical situations. We can expect therefore an accelerated uptake of WSN technology into the ERP market in the near future.

Requirements and challenges. Seamless integration is the key requirement for making WSNs in enterprise systems a success story. ERP vendors strive for generic, if possible universal, solutions. Consequently, WSNs have to shift from seeking ultimate efficiency through proprietary techniques to providing *standardized tools* for data collection, interfacing and deployment. Furthermore, the uptake of WSNs is challenged by cost concerns, packaging issues and battery lifetime, always projected to the competing RFID systems. In order to become a true success, *WSN technology must not compete with, but complement RFID* and thus deliver a combined solution.

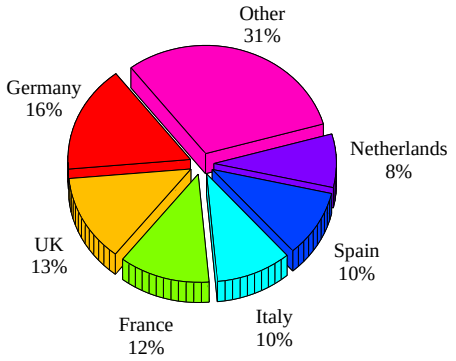


Figure 2.2: Western European logistics markets in 2006 (source DEGI Eurohypo [145]).

2.3 Transport and Logistics

Definition, facts and figures. The transport and logistics sector plays a major role in the world's economy. As a business concept, transport and logistics cover the flow and storage of materials from the point of origin to the point of consumption, including inventory management, transport, warehousing and distribution activities [174].

In Europe, the total turnover of the logistics sector in 2006 was estimated at € 800-900 billion. The total investment volume exceeded € 10 billion, which is more than double compared to year 2000. The long-term growth rate of the logistics industry is between 4% and 8%. About half of the European logistics industry is concentrated in only three countries: Germany, the United Kingdom and France [145] (see Figure 2.2).

In China, the aggregated turnover of logistics industry totaled about € 6 trillion in 2006, with more than 20 billion tons of goods transported. This means a growth rate of 15.3% year-on-year [72, 77]. China's transport and logistics sector employs more than 11 million people.

WSN technology perspectives. Transport and logistics applications often require specific monitoring procedures, especially in the case of perishable goods. Widely accepted as an efficient monitoring technology, WSNs qualify as a promising solution for such applications. Immediate use cases include cold chain management and check-and-trace processes [153]. In addition, the collaborative nature of WSNs opens perspectives for novel functionality, such as

verifying automatically the items loaded to be transported [129] or solving the RFID reader collision problem [127].

However, the greatest obstacle WSNs have to face is the relatively low market awareness. The situation will most likely change in the near future, as WSNs become more and more often associated with the active RFID technology. Market forecasts are therefore optimistic: active RFID and sensor networks would rise from 12.7% of the total RFID business in 2007 to 26.3% in 2017, meaning a US$ 7 billion market [33]. The exact share that WSN solution providers will get from this market remains an open question.

Requirements and challenges. It is widely accepted that Real Time Location Systems (RTLS) drive the penetration of WSN-based solutions in transport and logistics. As a consequence, *precise and fine-grained localization* is a hot research topic. Most localization algorithms rely on lateration techniques and strive for accurate 2-D or 3-D position estimation. In reality however, the end users often need a different type of information, such as "item X is on shelf Y" or "containers A, B and C are in the truck passing the exit gate right now". With respect to this kind of functionality, there is much unexplored potential in WSNs, as they can *combine intelligently relative distance information with inertial movement sensor data*. [1]

Two additional practical challenges concern the *density* and *packaging*. The first one is likely to produce major interference problems. In an example scenario of a typical distribution center, the number of sensor nodes can amount to tens of thousands in a relatively small area [128]. These figures would pose serious challenges to WSN communication protocols, especially to the MAC layer. The latter problem – packaging – urges the WSN community not only to accelerate size miniaturization, but also to deliver viable solutions for *disposable sensors nodes*, a far from trivial issue when having batteries on board.

2.4 Industrial Automation

Definition, facts and figures. Industrial processes are nowadays highly automated using production machines and robotic installations. An industrial automation system is typically organized hierarchically into four levels: the *plant*, *area*, *cell* and *field* level [73]. The top level (plant level) collects all the

[1] Relative distance estimation methods use hop count, signal strength (RSSI), time of arrival (ToA), time difference of arrival (TDoA), angle of arrival (AoA). Inertial movement information can be obtained with relatively insignificant costs from tilt switches and MEMS accelerometers.

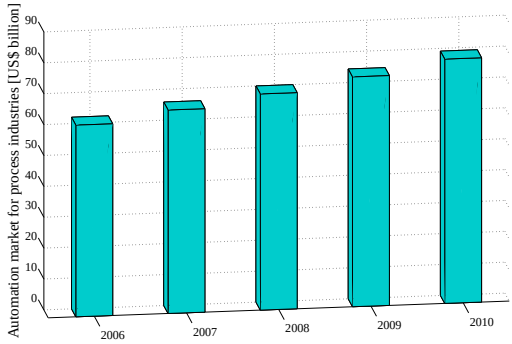


Figure 2.3: Estimate of industrial automation market evolution (source ARC Advisory Group [42]).

data and supervises the entire process by means of a Supervisory Control and Data Acquisition (SCADA) system [175]. The lowest level (field or device level) integrates Programmable Logic Controllers (PLC), sensors and actuators, being thus responsible for the direct data acquisition and process control.

The worldwide market for industrial automation systems was estimated at more than US$ 60 billion in 2006 and is forecast to continue on a solid growth path [42] (see Figure 2.3). China, India and the rest of Asia are driving the growth, with large infrastructural projects continuing to be booked. The Middle East also keeps on its capital investment boom, expanding the range of opportunities beyond the oil and gas and refining industries. North America and Western Europe are experiencing less growth, but are driven significantly by the need to modernize a rapidly aging automation infrastructure [137].

In recent years, Fieldbus systems [12, 25] have taken the leading position within the automation domain, with respect to the number of nodes installed (over 25 million [134]) and the standardization efforts. The global market for Fieldbus solutions reached more than US$ 800 million in 2006. Industrial Ethernet systems have also gained increasing popularity, benefiting from the ubiquitous usage of Ethernet devices in traditional networks. The global market for Industrial Ethernet was estimated at US$ 260 million in 2006 and predicted to grow at a 29% annual pace, reaching US$ 955 million in 2011 [40].

WSN technology perspectives. The primary reason for deploying WSNs is to reduce at least 50% of the costs associated with wiring. [2] This means

[2]Wiring installation costs are estimated at US$ 5 to $ 10 per foot [141].

that industrial wireless control systems represent a huge market opportunity. Although skepticism about their reliability and security does persist, market leaders such as ABB, ALSOM, Eaton, Honeywell, Invensys and General Electric are all investing in WSNs, with more OEMs expected to soon join in. Experts predict that on the medium term much more WSNs will be embedded with PLCs and plugged into industrial processes [60]. The most likely scenarios to gain initial traction will answer the following problems: (1) large industrial areas to cover, (2) need for enhanced automation and intercommunication, (3) high expense of equipment failure and (4) energy-intensive equipment.

Requirements and challenges. Packet loss, time delay and low data rates associated with wireless communications can significantly degrade the performance of the overall control system. Therefore, WSN community has to focus initially on *increasing the reliability* of the network protocol stack to the level required by industrial automation applications [141, 178]. Secondary, but still important, properties that need attention are *real-timeness* and *security* [178].

It is unrealistic at the moment to assume that wireless communication in general and WSNs in particular will replace the already in-place wired systems. Instead, hybrid wired/wireless schemes are envisioned to be put into practice in the near future. This combination, although definitely increases the overall flexibility, brings about two additional challenges: the design of the *coupling devices* (repeaters, bridges, gateways) and the *delay* (potentially even bottleneck) introduced by these coupling devices [178].

2.5 Safety-Critical Processes

Definition, facts and figures. The process safety system market is directly linked to industrial fields with high potential risks, such as fire and gas monitoring, oil and gas, chemical industry, nuclear electrical power. As a result, increased worldwide energy demands led to a solid rate of growth for safety instrumented systems. The global market reached more than US$ 1 billion in 2006 (see Figure 2.4), largely due to the economic development of China and India fueling investments in oil and gas production and refining. Other factors contributing to the positive trend of the safety system market include: (1) adoption of safety standards, (2) increased cost of accidents and insurance, (3) strict environmental regulations and (4) obsolescence of older generation installed safety systems [41].

WSN technology perspectives. Safety-critical processes are a promising target market for WSNs. Large industrial corporations tighten their risk

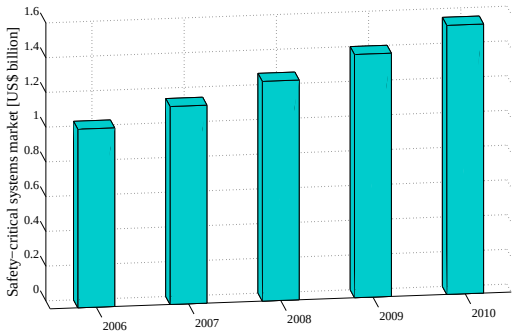


Figure 2.4: Estimate of safety-critical systems market evolution (source ARC Advisory Group [41]).

tolerance and put forward the concept "safety is good business". For implementing this concept, WSN technology appears more than appropriate because it offers a low cost, distributed, reactive and easy-to-manage solution. British Petroleum (BP) for example conducted successful application trials at a chemical plant using the WSN technology developed in the EU-funded project CoBIs [127, 4]. Such initiatives are not singular and several projects study the potential usage of WSNs in fire detection, firefighter assistance and disaster management [2, 13, 26].

Requirements and challenges. The typical usage of WSNs in safety-critical processes implies long term monitoring and event detection. This means that network lifetime and quality of event detection are equally important problems. As a consequence, WSNs must implement *intelligent and distributed inference methods* that can guarantee, even at low duty cycles, *fast detection times* as well as *low false alarm rates*.

Further challenges are to be expected at deployment time. When dealing with safety-critical applications, the costs of intrinsically safe equipment adds a high factor concerning *packaging and quality control*. Even if the WSN technology is ubiquitous and cheap, these costs remain rather constant, since strict guidelines have to be fulfilled, such as the Directive 94/9/EC (Equipment intended for use in potentially explosive atmospheres – ATEX) [10].

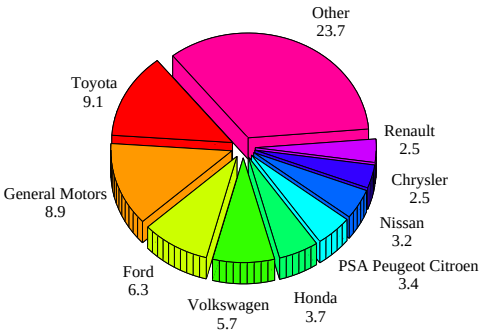


Figure 2.5: World motor vehicle production (in million units) by manufacturer in 2006 (source OICA [22]).

2.6 Automotive Industry

Definition, facts and figures. The automotive industry represents a key driver of global economy, employing 8 million people (over 5% of the world's total manufacturing employment) only in producing motor vehicles and more than 50 million people in related manufacturing and service provision. In 2006, more than 69 million motor vehicles were produced worldwide (see Figure 2.5 for the major manufacturers). The total turnover was estimated at approximately € 2 trillion [22]. In 2007, the markets in USA, Canada, Western Europe and Japan were stagnating, while those in South America, Eastern Europe and India were on a positive trend [172].

Today, an automotive system consists of several subsystems, which are further divided in Electrical Control Units (ECUs). For a modern car, this adds up to 70 ECUs that control altogether more than 2500 variables and signals [111, 36]. Networked communication is an essential ingredient in managing this distributed control system. The most important automotive subsystems that rely heavily on networking are: (1) chassis, including Electronic Stability Program (ESP) and Anti-lock Brake System (ABS), (2) air-bags, (3) powertrain, including engine control, (4) comfort electronics, including climate control and cruise control, (5) x-by-wire systems [3] and (6) multimedia and infotaiment [135].

WSN technology perspectives. In-vehicle networks of sensors and ECUs

[3] X-by-wire denotes the new subsystems replacing hydraulics and mechanics with electronics. Examples include steer-by-wire, shift-by-wire and break-by-wire.

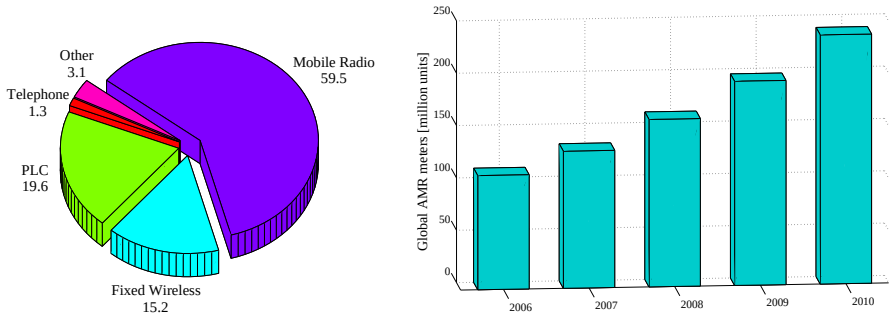
are currently wired. The most widely used technologies are Controller Area Network (CAN) [5] (standardized in the early 90's), Media Oriented Systems Transport (MOST) [17] and FlexRay [14]. There are, however, several incentives for the adoption of wireless communication in the automotive industry. With the continuously growing number of sensor and electronics embedded in cars, the wired architectures are soon prone to scalability problems [110]. WSNs can offer a better alternative by implementing local control loops and providing a flexible communication mechanism. Moreover, inter-vehicle and vehicle-to-roadside communication creates the premises for increased passenger safety and comfort. Again, WSN technology forms a solid basis for implementing such Vehicular Ad-Hoc Networks (VANETs) [176] and building the future automated highway [84].

Requirements and challenges. Automotive communication in general has to meet high *reliability* (in presence of communication errors and system faults) and *determinism* (strict timeliness guarantees, for example when the airbag has to be inflated) requirements [135]. While the first is a constant topic of interest for the WSN community, the latter may be a serious challenge for wireless communication. The feasibility of in-vehicle WSNs is currently studied experimentally. Recent results showed that the wireless channel can satisfy a maximum packet delay requirement of less than 500 ms and a packet reception rate of 98% [163]. However, both node placement and antenna orientation have a significant impact on communication performance parameters [101].

Considering inter-vehicle and especially vehicle-to-roadside interaction, the biggest issue is *mobility*. For this purpose, typical WSN communication protocols have to be tailored for high-speed topology changes and have to implement fast handover mechanisms.

2.7 Automatic Meter Reading

Definition, facts and figures. Automatic Meter Reading (AMR) technology enables the remote collection of consumption data from utility meters using telephony, fixed wireless, mobile radio, power-line and satellite communications. The added value is more than just reducing the cost of manually reading meters. Utility providers have more control over energy, power or water delivery, as well as payment. Recent legislative pressure, such as the Energy Act of 2005 [9] and the Clean Water Act [3], creates the incentives for utilities to introduce real-time



(a) (b)

Figure 2.6: (a) US AMR meters by technology (million units); (b) Estimate of global AMR meters deployments (source ON World [61]).

pricing, time of use (TOU) metering and submetering [4] for consumers.

Today, AMR is a mature market, estimated to reach US$ 1,7 billion in 2010. North America makes up 80% of the total AMR market, with approximately 100 million AMR meters deployed (see Figure 2.6(a)). However, we currently witness an accelerated adoption of AMR solutions also in Europe, most notably for electricity consumers. Manual reading of more than 200 million electricity meters in Europe costs about US$ 5,7 billion anually, but still fails to produce accurate energy bills. The global market is therefore envisioned to grow steadily and reach more than 230 million utility AMR units installed [61], as shown in Figure 2.6(b).

WSN technology perspectives. Utilities expect savings of approximately US$ 3 per meter per month by implementing advanced AMR and meter management [61]. Wireless technology is the most appealing solution because it minimizes the wiring and monthly operating costs. In addition, utilities would have their own wireless networks and thus have complete control when reconfiguring or expanding meter reading territory.

ZigBee [31] is currently seen as a potential standard solution for advanced AMR and especially submetering. The most significant issues concern the relatively short range and privacy and security. Market forecasts are therefore reserved, predicting ZigBee to make up about 26% of all US fixed wireless AMR units by 2010 [61].

[4] Utility submetering allows a landlord or other multi-tenant property to bill tenants for individual measured utility usage.

Coronis Systems [6] reported some of the largest WSN-based AMR installations, consisting of 30,000 wireless nodes in Europe and 50,000 nodes in China. Their proprietary Wavenis platform is going to become an open specification basis for industry-wide standardization.

As the global energy demand boosts with unprecedented pace [32], utilities and subsequently AMR market are expected to move fast toward implementing large-scale systems for better energy management and saving. The uptake of WSN technology is therefore conditioned by how quickly it can deliver a robust and complete solution for such infrastructures.

Requirements and challenges. AMR is a niche market for WSNs, with special characteristics to be considered. The most peculiar example is probably the case of electric utilities, where the foremost WSN challenge – energy consumption – is no longer a critical aspect. Instead, WSNs must solve different problems. *Two-way communication* is utterly important, as utilities will invest into a complete solution that enables them not only to collect consumption data centrally, but also to poll on-demand and manage the meters remotely. Although the WSN community studied extensively the sources-to-sink communication paradigm, there is little work concerning the opposite way of interaction. *Security and privacy* are to be enforced on both directions of communication. Furthermore, *scalability* is a practical problem of orders of magnitude higher than current WSN deployments, if we consider each residential meter equipped with a wireless sensor node. To tackle the scalability problem, the WSN protocol stack should exploit the static, regular nature of the deployments, and thus *optimize the communication flow for hierarchic cluster-tree network topologies.*

2.8 Discussion

This study would be incomplete without a more general discussion on the complex process of technology adoption. As any other innovation, WSN technology is subjected to a *diffusion* process, which was defined by Rogers as "the process by which an innovation is communicated through certain channels over a period of time among the members of a social system" [147]. The diffusion of innovation follows an S-curve, in which we can distinguish five adopter categories: innovators, early adopters, early majority, late majority, and laggards. These categories typically follow a normal distribution, with few innovators and early adopters (around 16%) selecting the new technology in the beginning, the early majority and late majority making for 68%, and finally the laggards constituting the remaining 16%. From the adopter perspective, the diffusion process evolves

over time through five stages: *knowledge, persuasion, decision, implementation* and *confirmation*. The final success or failure of an innovation depends on a multitude of factors, but an essential aspect is the speed of adoption (both take-off and later growth), in other words how steep the adoption S-curve is and how fast the adopters pass through the five stages, from knowledge to implementation and confirmation.

Where does WSN technology stand in this frame? The market examples presented in the previous sections indicate that WSNs climb the S-curve from innovators to early adopters. However, the diffusion process does not evolve uniformly: many potential users are still in the phase of acquiring knowledge on the technology, while others have already made the implementation decisions. To better understand these differences, we have to consider that WSNs had a relatively short trajectory from visionary research to products. Consequently, the current level of awareness on their potential varies significantly within various application domains and within different industrial organizations. As shown by Waarts et al. [166], the perceived potential value of the new technology and the level of industry competitiveness drive early adoption to a higher extent than late adoption, while the latter is influenced more by implementation issues, such as compatibility with current procedures, reliability, scalability, etc.

The potentials of WSNs have to be analyzed from multiple points of view. So far, we have identified certain strengths and opportunities with respect to the surveyed market domains, but what about weaknesses and threats? Although this study does not aim at pursuing a detailed SWOT [5] or PEST [6] analysis, we can still outline the most important adverse factors in the WSN adoption process. The technological weaknesses have already been named in the introduction from Chapter 1: extreme resource limitations, unreliable wireless communication, harsh operating conditions. The external factors (or threats, according to SWOT terminology) are related to three broad categories:

- *Regulation*, which can increase the already substantial technological difficulties (for example, the limited available frequency spectrum imposed by legislation) or add up to the production costs and thus cancel the "low-

[5] SWOT is an acronym for "Strengths, Weaknesses, Opportunities and Threats". SWOT analysis is a tool for evaluating a strategy, project, business venture, or any other idea, by identifying the strengths and weaknesses of the organization, as well as the external opportunities and threats. The technique is credited to Albert Humphrey.

[6] PEST stands for "Political, Economic, Social and Technological". PEST analysis gives an overview of the different macroenvironmental factors that an organization has to take into consideration when studying market growth or decline, business position, potential and direction for operations.

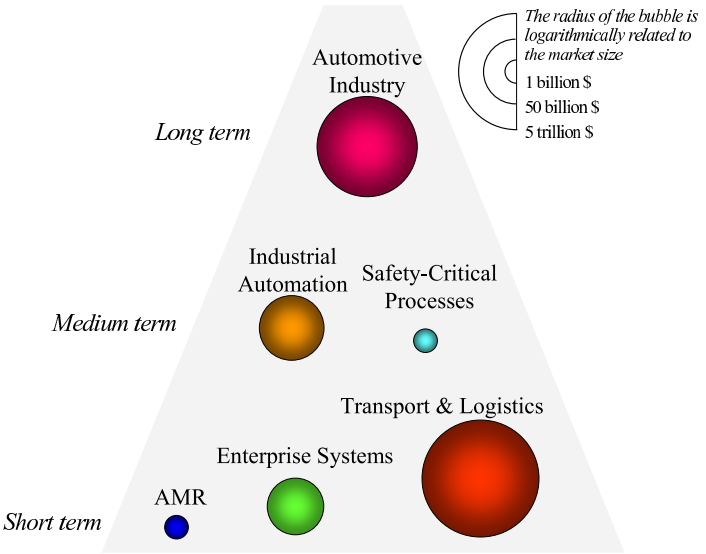


Figure 2.7: Summary of market situation and perspectives for WSN adoption.

price" distinctive feature of WSNs (for example, packaging norms related to equipment installed on safety-critical sites).

- *Competing technologies*, with RFID being the prominent example. As already mentioned in this chapter, starting from the fact that RFID passed the technology adoption test and is currently a well-established paradigm, WSNs should find their place *next to* and not *instead of* RFIDs on the market.
- *Society*, along with issues related to usability, user acceptance and, above all, privacy concerns and the danger of malicious use of the new technology.

2.9 Conclusions

We analyzed in this chapter the WSN market perspective in six industrial and business application domains: ERP systems, transport and logistics, industrial

automation, safety-critical processes, automotive industry and AMR. Although generally enthusiast about wireless in general and WSNs in particular, the market is still reserved in adopting WSN-based solutions on a large scale. Quantitative predictions on this topic are merely speculative, since taking WSNs from the research lab to real life is still a far from trivial task.

Figure 2.7 summarizes the market situation and the most likely perspectives of WSNs on the short, medium and long term. Judging by the total market size and dynamics, transport and logistics is the most attractive domain at the moment. Even if dominated by RFID-based solutions, the transport and logistics market can represent for WSNs what venture capitalists call "a small piece of a large pie" [144]. Automotive industry falls under the same category, but it is realistic to assume in this case a much slower shift from wired to wireless systems. On the short term, ERP and AMR markets are particularly promising. Concerning the latter, however, it is debatable whether WSN producers will provide the solutions or utilities will develop their custom wireless infrastructures. Industrial automation and system safety are perhaps the best-match application domains for WSNs. The adoption of sensor networks cannot be expected on the short term though, due to the high reliability and safety requirements involved in practice.

Chapter 3

Service-Oriented Architecture

In this chapter, we present a three-layer service-oriented architecture that accommodates different sensor platforms and exposes their functionality in a uniform way to the business application. The ultimate goal is to delegate well-defined parts of the business logic to the low-cost embedded devices, and thus reduce the process execution costs and improve the response time in safety-critical situations. We present both practical tests and application trials that confirm the feasibility of our solution and the benefits of multi-platform integration. This chapter is largely based on the article "Decentralized Enterprise Systems: A Multiplatform Wireless Sensor Network Approach" in IEEE Wireless Communications Magazine, Vol. 14(6), December 2007 [127], which is joint work with N. Meratnia, P. Havinga, L. M. Sa De Souza, J. Muller, P. Spiess, S. Haller, T. Riedel, C. Decker and G. Stromberg. This work has been partially sponsored by the European Commission as part of the CoBIs project (IST-004270).

3.1 Introduction

Sensing and actuating represent nowadays major functionality when describing the vision of pervasive computing. Made possible by the proliferation of wireless technologies and the advances in manufacturing low-cost, low-power devices, massively deployed wireless sensor and actuator networks (WSN and WSAN) [34] target a large number of applications, ranging from smart spaces [28] to industrial processes [4], and even planetary sensing or space exploration [80]. The enthusiasm generated by these countless possibilities has led in recent years to an outbreak of diverse sensor network platforms, covering both the hardware and the software. Without claiming to give a complete taxonomy, we can identify the following three broad classes of sensor nodes currently in development:

Class 1 – Tiny, cheap, energy-constrained, numerous devices, illustrating the vision of Smart Dust [100]. Application domains include environmental monitoring, battlefield and logistic processes.

Class 2 – Multi-functional, user-centric, rechargeable devices, covering health care, games and sports, as well as various mobile applications [74].

Class 3 – Powerful, reliable devices, approaching the capabilities of an embedded computer [102] and targeting applications with strict requirements such as industry and military.

Today's users are easily overwhelmed by the number of options available when they have to choose the right technology for their application. Due to the unique challenges of WSN [63], the platforms are typically specialized for specific purposes (e.g. data collection, target tracking), so it is often the case that complex applications require the combination of multiple proprietary technologies and customized platforms. As a result, the users are confronted with a considerable amount of low-level programming, tuning and tedious testing. Furthermore, the management, monitoring and administration of a system with highly distributed logic is a very complex task. Without the right tools and architecture, it can increase the total cost of ownership to a point where the deployment of this technology becomes commercially uninteresting.

A service-oriented architecture (SOA) is helpful in solving these issues. The integration efforts are minimized by hiding much of the implementation details and exposing only the functionality of the WSN in use. The management is also simplified because the logic is encapsulated in services with a manageable granularity. The services can be deployed, removed or upgraded from a central location in order to adapt the system to the business needs.

In this chapter, we focus on the integration of various WSN platforms in large-scale enterprise environments. Service-oriented architectures based on

Figure 3.1: Application trial in Hull, UK: Chemical containers stored in a warehouse (left); Detail with a sensor node attached to a container (right).

Web Services technology have recently become popular for building complex yet flexible enterprise systems [180]. However, taking the SOA concept to the level of distributed embedded devices represents an intricate problem [50]. Even if sensor nodes of Class 3 may have the necessary resources, the ones belonging to Classes 1 and 2 are clearly too limited for such a complex task. Consequently, the efforts of the WSN community in this direction are still incipient and are focused on small-scale settings.

In what follows we propose a three-layer SOA designed for heterogeneous WSNs. Motivated by real business cases identified at BP premises in UK, we utilize three sensor platforms (see Sec. 3.3) specialized on specific tasks: dynamic networking under mobility conditions, large-scale infrastructure and coexistence with RFID. In order to leverage the effort of the resource-constrained sensor nodes (i.e. Classes 1 and 2), the platform gateways expose the WSN functionality to the business applications in a uniform way, i.e. by using the Universal Plug and Play (UPnP) standard. The high-level business logic and management are implemented using the SAP Web Application Server (WebAS) and incorporated within the SAP enterprise software.

The main contribution of this work is therefore to show the feasibility of a complete system - from the top business application to the bottom sensor nodes - which accommodates and uses heterogeneous hardware and software resources in a uniform way.

3.2 Application Scenario

The application scenario that motivated this work comes from the oil&gas industry. Every year, the U.S. Department of Labor registers numerous occupational accidents in this field (e.g. only in 2004, there have been recorded 52,830 non-fatal injuries and 29 fatalities due to the exposure to harmful substances or environments). WSN represent a viable solution to this problem. Sensor nodes can collaboratively determine potential hazardous situations, and alert or take an action at the point of interest [159]. In addition, the combination of WSN and RFID technology can prevent errors in the manipulation and storage of chemical containers, leading thus to increased safety and reduced logistic costs. In the EU-funded project CoBIs (Collaborative Business Items) [4], we extend these ideas by designing and implementing a distributed, service-oriented enterprise system, which incorporates the latest advances in WSN technology. The field tests were carried out at a chemical plant of BP in Hull, UK (see Fig. 3.1), where the following use cases had been identified:

Storage and manipulation of hazardous substances. Chemical containers storing reactive substances must be handled according to a strict list of safety regulations. The following situations are to be avoided: *(i)* storing incompatible substances in close proximity of each other, *(ii)* exceeding the maximum storage volume threshold for any hazardous substance and *(iii)* storing hazardous substances in temporary, unprotected areas longer than a specific maximum time period.

Continuous monitoring of environmental conditions. At the site of a chemical plant, environmental parameters, such as temperature, humidity and light, should be continuously monitored and abnormal conditions should trigger immediate alarms and local actions.

Smart shelves in warehouses with chemical containers. RFID readers placed on the shelves of the warehouses can improve significantly the tracking and identification of various objects (chemical containers, tools, etc.), already outfitted with RFID tags. To overcome the major problem of RFID reader interference, sensor nodes attached to the readers can agree on a non-overlapping subset of readers that are switched on.

These three use cases led us to an integrated approach, since there was no single WSN platform that could optimally fulfill all the tasks. Consequently, we opted for using three sensor platforms, namely *Particle* (produced by Particle Computer), *µNode* (produced by Ambient Systems) and *Sindrion* (produced by Infineon Technologies). The prototype sensor nodes are shown in Fig. 3.2. In the following section, we give a brief technical overview of the three platforms,

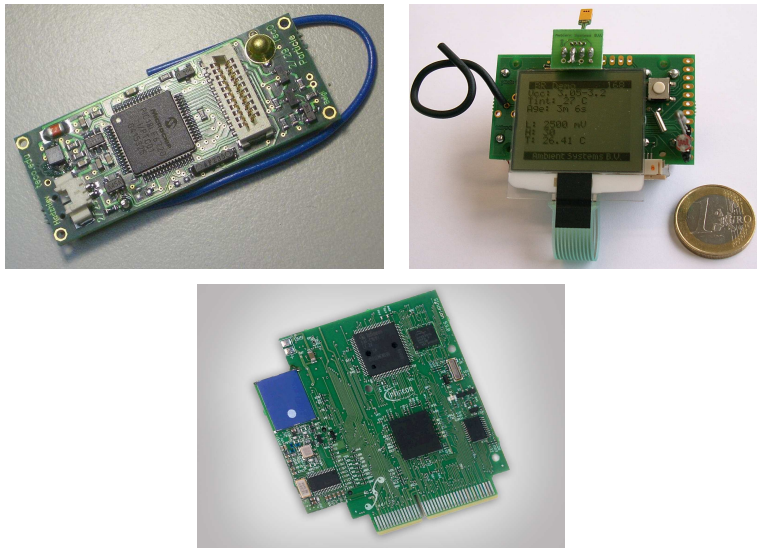


Figure 3.2: The three sensor platforms: Particle, μNode and Sindrion.

indicating the specific services that each of them delivers within the given scenario.

3.3 Sensor Network Platforms

The Particle node [23] comprises a communication board with the PIC18f6720 microcontroller and TR1001 transceiver. Various types of sensors can be attached to the communication board. The wireless communication uses the AwareCon protocol [43], which is designed to handle high mobility and density of nodes. This makes the Particle platform well suited for equipping chemical containers handled by human operators and checking potential dangerous situations, as described in use case 1 of our scenario.

The μNode platform [1] represents a low-power, general purpose sensor node, built around the MSP430 microcontroller and a single-chip radio transceiver for the 433/868/915 MHz ISM band. After deployment, the μNodes self-organize into a multihop network, through which data can be routed back and forth to a designated sink node. This platform is ideal for building large scale sens-

ing infrastructures, which can function unattended for long periods of time. Since many chemicals must be stored under specific ambient conditions, we use the μNode sensors for continuously monitoring environmental conditions, as described in use case 2 of our scenario.

The Sindrion platform [83] comes with native support of standard networking protocols (e.g. DHCP, IP, UDP, TCP, HTTP) on the sensor nodes. The nodes comprise an Infineon TDA 5250 868MHz transceiver and a 16-bit Infineon microcontroller. The Sindrion nodes integrate natively into the IT infrastructure by a network adapter attached as a USB dongle. In contrast to the other platforms, the Sindrion nodes feature standard UPnP discovery and description. In addition, the Sindrion nodes are designed to be tightly coupled to RFID readers. The combination represents a smart ISO15693 RFID reader, which negotiates the access to the wireless medium in order to prevent interferences and exposes itself in the network as a full-blown UPnP device providing service-based functionality. Consequently, this system implements the "smart shelves" concept, as described in use case 3 of our scenario.

3.4 Architecture Overview

In this section, we define a three-layer SOA, which is intended to provide a bridge between the business applications and the underlying sensor and actuator networks. The three layers – the back-end, the gateway and the front-end layers – are illustrated in Fig.3.3 and discussed in detail throughout the following subsections.

3.4.1 The Back-end (or Application) Layer

The business applications should be able to access the services offered by the WSN at the level of Web Services. In order to achieve this goal, the back-end layer benefits from the uniform interfaces offered by the gateway layer. The components of the back-end layer are therefore implemented completely independent from the underlying platforms. As shown in Fig.3.3, we distinguish the following high-level components:

Service Repository – contains a database of the available services, including their description and implementation. A single service can have several implementations (or executables) on different platforms. The service descriptions are XML documents formatted in CoBIL (an acronym for CoBIs Language, see [4] for the XML schema). A CoBIL description starts with the service in-

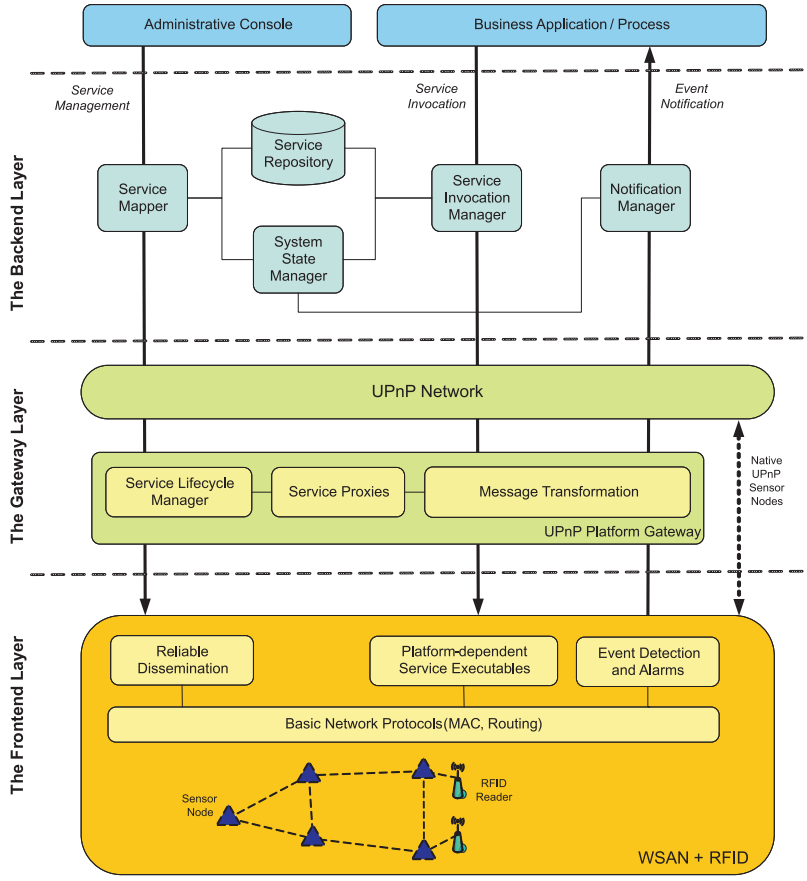


Figure 3.3: Three-layer service-oriented architecture.

terface, i.e. the set of operations that all the service executables offer and the events they generate. The interface definition is derived from WSDL (Web Service Description Language). For each executable, the CoBIL description contains specific information, such as the reference point, the target platform, etc. In the end, the technical requirements to be used at deployment time are specified (e.g. which sensor and actuators the target nodes have to feature, the minimum remaining energy, the necessary network bandwidth, etc.).

System State Manager – stores the operational state of the nodes, such

as the services running, resources available, current position information, etc.

Service Mapper – performs the systematic mapping of the services to the nodes, based on the service descriptions (technical requirements, composition constraints) and the system state. The Service Mapper is used during service deployment (see Sec. 3.5.1).

Service Invocation Manager – processes the service invocations issued by the business application running on the back-end. More specifically, the Service Invocation Manager contacts the node(s) executing the specified service and retrieves the results of the service invocation back to application.

Notification Manager – implements a web service interface for distribution of event messages. Interested client applications can register with the Notification Manager and will receive notifications for all relevant event messages.

3.4.2 The Gateway (or Platform Abstraction) Layer

The gateway layer has an essential role in harmonizing different sensor platforms. We opted for the UPnP standard as the uniform interface between the application layer and the underlying WSN. In recent years, UPnP has been widely accepted as a simple and robust standard for ad-hoc and unmanaged networks. Being designed to support zero-configuration and automatic discovery for a breadth of devices from different vendors, UPnP facilitates the integration of new platforms via simple standardized mechanisms. In our case, the Sindrion platform already implements a basic UPnP interface, which makes it capable of connecting directly to the back-end layer. However, the Particle and μNode platforms are too resource constrained to run natively UPnP. It is therefore the responsibility of the gateway layer to handle the proprietary WSN mechanisms and expose the service-oriented functionality through a standard UPnP interface.

With respect to our reference architecture, the gateway layer provides the following functionality:

Message Transformation – handles the packet-level translation between the proprietary WSN messages and UPnP arguments.

Service Lifecycle Manager – assists the deployment of new services in the network. Deployment requests are issued to the gateway by specifying the service executable and the XML-based deployment description.

The key feature of the gateway is the dynamic instantiation of service proxies. Service proxies can be accessed like native UPnP devices, providing detailed service descriptions for the implemented functionality. However, the service prox-

ies only exist as virtual representations of the service interfaces. The gateway transforms the requests issued to the service proxies into WSN messages and vice-versa. In addition, UPnP handles service discovery natively once a proxy is initiated. This means that service proxies have to be instantiated whenever a new service is provided by the WSN and destructed when the service becomes unavailable.

3.4.3 The Front-end (or Device) Layer

The device layer encompasses the multitude of WSN and RFID technologies. The design guidelines are driven by the requirements of the industrial applications and the typical constraints in terms of energy, bandwidth, computational power and storage. As a result, the components developed within the device layer have to meet the following objectives: (1) energy efficiency (both at the node level and the network level), (2) responsiveness, (3) reliability, (4) scalability, (5) reconfigurability and (6) usability. In Fig. 3.3, we highlight the most relevant components with respect to the service integration:

Reliable Dissemination – enables the service deployment and updating. It works on top of the unreliable WSN communication protocols and guarantees that the new service executables are transferred correctly to the target nodes. Due to the heterogeneity of the deployed WSN, the reliable dissemination provides multicast support for addressing selectively groups of nodes, being thus more scalable than unicast or flood-based solutions (see Sec. 3.6.2). In addition, it works in conjunction with the MAC layer in order to reduce the communication and idle listening to a minimum. From the design perspective, the reliable dissemination component fulfills therefore objectives 1, 3, 4 and 5.

Platform-dependent Service Executables – represent running instances of the services, which can be invoked by the business application in order to execute specific tasks or retrieve a certain information. The platform-dependent services that we developed include synchronization, proximity, localization, data aggregation and querying. With respect to the design guidelines, these services relate mainly to objectives 1, 4 and 6.

Event Detection and Alarms – target timely and reliable detection of special situations that should be reported to the central system. In addition, the sensor nodes may signal and handle locally these situations, for increasing the overall responsiveness. Compact rule engines (see Sec. 3.6.1) are shown to be an effective solution to objectives 2 and 6, as they involve minor overhead at runtime while facilitating significantly the task of the business application developer.

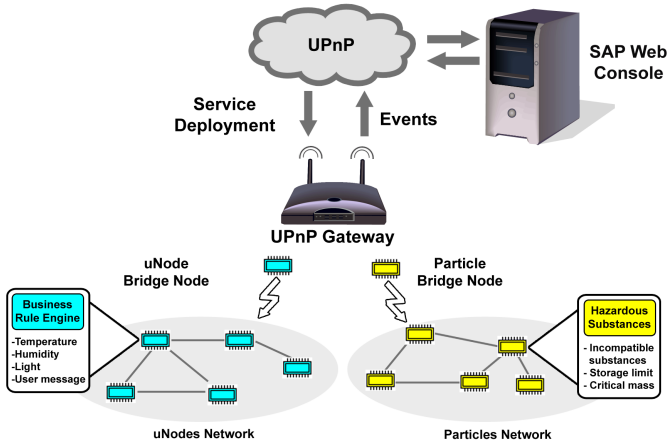


Figure 3.4: The complete system using a hybrid setting with both Particle nodes and μNodes.

Basic Network Protocols – form the platform-specific communication stack. Considerable importance is given to the MAC protocols, which contribute directly to the fulfillment of objectives 1, 2, 3 and 4.

For the detailed design of all the components within the device layer, as well as the performance evaluation and implementation results, the reader is referred to the public deliverables of the CoBIs project [4].

3.5 Functional Overview

This section gives a functional overview of the entire system. More specifically, the three main operations – service lifecycle management, service invocation and event notification – are explained with respect to the reference architecture (see the thick arrow flows in Fig. 3.3).

3.5.1 Service Lifecycle Management

Service Lifecycle Management is responsible for three major administrative tasks: the deployment, update and removal of services. For the sake of brevity, we describe only the service deployment; the other two operations bear clear similarities in their work flows. The deployment of a new service is initiated

by the administrator command in the Administrative Console running on the back-end. Firstly, the Service Mapper creates a service mapping, based on the current state of the system (retrieved from the System State Manager) and the technical requirements specified in the CoBIL description of the service (from the Service Repository). As a result, the nodes (identified by individual IDs, group ID or just a geographical area) where to instantiate the service are chosen. Secondly, the deployment request is sent to the Service Lifecycle Manager counterpart on the UPnP gateway. The gateway uses the URI (Uniform Resource Identifier) of the service executable and the executable-specific information from the service description in order to launch the platform-dependent deployment. Thirdly, the new executable is transferred to the WSN, in our case using a reliable multicast dissemination protocol (see Sec. 3.6.2). Finally, the result of the deployment (success/error) is sent back to the Administrative Console. The log of the dissemination session in the WSN is stored on the gateway for post-analysis purposes.

3.5.2 Service Invocation

Service invocation is the key operation through which the business applications running on the back-end use the functionality of the WSN. To initiate this process, the business application issues a service invocation request to the Service Invocation Manager, which in turn retrieves the service description from the Service Repository, in order to validate the parameters and check the service requirements. After identifying the nodes that offer the service by consulting the System State Manager, the service invocation request is delivered to the gateway layer. On the gateway, the UPnP - WSN transformation takes place. For this purpose, a transformation description is associated with each UPnP service description. The descriptions are automatically parsed by the UPnP stack, which already provides the RPC (Remote Procedure Call) dispatching and eventing facilities. For specifying the transformations we use a simple template-based transformation mechanism that works bi-directionally. When a RPC call is received, each outgoing argument is serialized via the template-based transformation to the UPnP state variable and the assembled message is then sent to the WSN. The addressed nodes execute the service and issue a response message, or a timeout/error message. The response messages from the WSN are converted to the according arguments by inverting the template-based transformation process. In the Sindrion platform, however, application-specific proxies are optionally made available directly from the sensor nodes. In this case, the message transformation is not required, since the Sindrion programming envi-

ronment supports the native development of UPnP device representations.

3.5.3 Event Notification

An important functionality of the WSN is the ability to detect and signal events. In this case, the business applications represent event consumers, which subscribe to the Notification Manager component and provide a filter of the relevant events. The sensor nodes detecting the event send notification messages toward the gateway. After being routed in the WSN, the messages arrive at the gateway, which performs the WSN – UPnP transformation that leads to a UPnP state variable change. UPnP's General Event Notification Architecture (GENA) handles the events and transmits them to the Notification Manager. The Notification Manager selects the consumers that have subscribed for the specific events, and distributes them accordingly. In addition, it publishes the notifications to the System State Manager for updating the system operational state.

3.6 Implementation and Testing

The previous sections provided a general overview of the integrated SOA, abstracting from the underlying hardware and software platforms. In this section, we discuss the most relevant implementation details and present the operation of our system with a hybrid setting using both Particle and μNode sensor nodes (see Fig. 3.4). For brevity, we focus on these two platforms because they illustrate best the abstraction of the platform gateway, in contrast with the Sindrion platform, which is directly accessible through the standard UPnP mechanisms.

As gateway device, we utilize an off-the-shelf WLAN router (Asus WL-550g) running OpenWRT (Linux derivative for embedded devices), on which we implement the Cyberlink UPnP stack. The connection to the WSN communication protocols is achieved by connecting both Particle and μNode bridge nodes to the USB ports of the router. In this way, we can build a cost-effective hybrid gateway that can communicate with both sensor network platforms.

The components of the back-end layer are implemented as Web Services and deployed on a SAP Web Application Server. They are developed in Java for portability, using SAP NetWeaver Developer Studio.

In the following, we present the experiments and application trials that we performed according to the scenario described in Sec. 3.2.

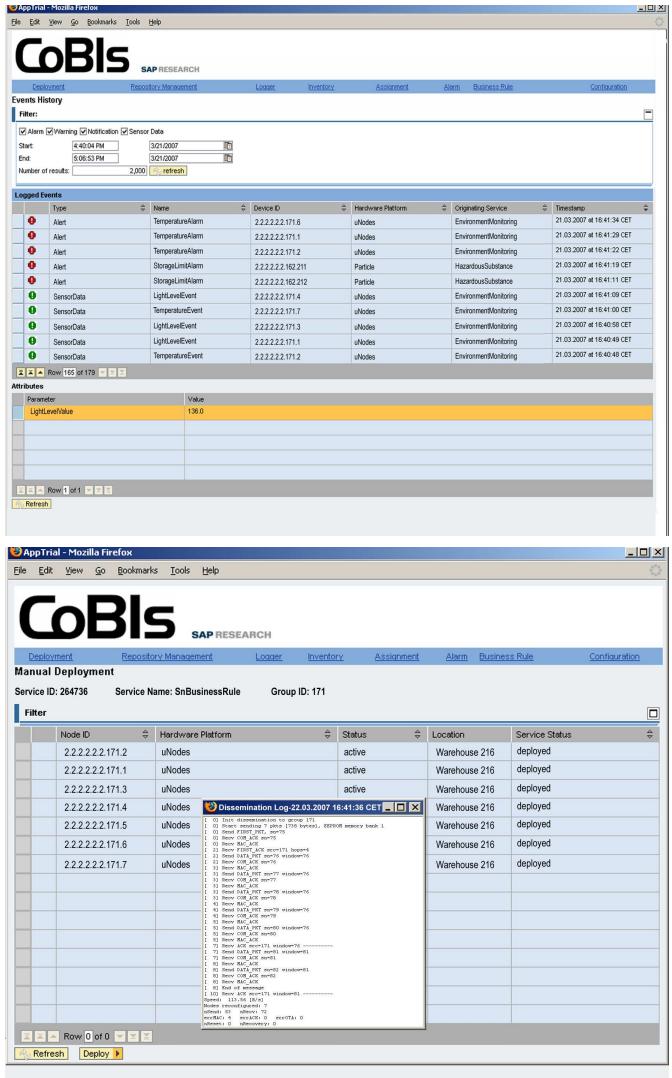


Figure 3.5: Web console: Events from both Particle nodes and μNodes (top); A new service is being deployed in the WSN (bottom).

3.6.1 Hazardous Substances and Environmental Monitoring

The first setting illustrates the use cases 1 and 2 of our scenario, namely handling hazardous substances and monitoring the environmental conditions. The first of these tasks is implemented on the Particle nodes and the latter on μNodes. Both platforms use the UPnP hybrid gateway to receive requests from the back-end and report the events generated within the network.

The Particle nodes execute the "hazardous substances" service in order to alert about situations that do not comply with the safety regulations given by the oil&gas company (BP in our case). The alarms are triggered locally by the nodes placed on the chemical containers and are reported to the back-end application (along with the location information). The overall responsiveness of the service has to be less than 2 seconds.

In our setting, the Particle nodes use the AwareCon [43] communication protocol, which is a TDMA-based protocol. The timeframe is divided in 70 slots, each of 13ms. The duty cycle is set to 35% (25 slots). Therefore, it can be guaranteed that the delay for detecting a hazardous situation is less than 2 seconds. In order to detect the situations where incompatible substances come in close proximity of each other, the Particle nodes are equipped with TSOP IR receivers and implement a diffuse infrared-based location method. The main advantages of this solution are the low signal processing overhead and the low power operation. The disadvantages concern the disturbances from occluding objects and direct sun light. Depending on these conditions, the accuracy lies between 30cm-1m. The "hazardous substances" service is loosely coupled to the location service and takes its input to compute storage incompatibilities and limits. Limits and incompatibility classes can be configured via the service update interface from the back-end system. The nodes update the storage status within the surrounding space by broadcasting their information during the wake phase of the AwareCon protocol. The alarms are signalled collectively by flashing visible LEDs, and reported to the back-end system. The resources needed on the Particle nodes are 746 bytes of FLASH memory and 10 bytes of RAM for the location service, and 8.5 kB of FLASH memory and 242 bytes of RAM for the "hazardous substances" service, respectively.

Monitoring the environmental conditions for potential dangerous situations is implemented through the business rules [125] support of the μNodes. The business rules for sensor nodes express simple business logic in a compact and efficient way, by following the execution chain *Observe – Check rules – Take action*. In our setting, the μNodes are equipped with the following: internal

voltage and temperature sensors, a light dependent resistor (LDR), a combined temperature and humidity sensor (SHT), and a push-button. A typical rule set for this setting amounts to approximately 10 rules (200 bytes), which means little network overhead in the reconfiguration phase. The rule engine is also very lightweight, with less than 1 kB code memory footprint. The average sensing driver size is $\approx$ 300 bytes, whereas the average action service module size is $\approx$ 130 bytes. In total, a typical business rule-based program requires only $\approx$ 3.5 kB, which represents 7.3% of the available FLASH memory. The detailed description and evaluation of the business rule engine can be found in Chapter 5.

As depicted in Fig. 3.5a, the integration of the Particle nodes and μNodes is successful. Events (hazardous substances and business rules alarms) from both platforms reach the back-end (a Web-based console in this case) through the UPnP gateway. Sensor readings, e.g. the light level, can also be obtained, by invoking the corresponding service. The integration process is completely transparent to the back-end layer.

3.6.2 Service Deployment

Dynamic reconfiguration of the sensor network at runtime directly supports the service deployment in our reference architecture (see Sec. 3.5.1). Due to the critical reliability and scalability requirements, a specialized protocol must be used within the WSN, which guarantees data delivery with minimal energy expenditure. We use RMD [123], a multicast-based dissemination protocol that supports reconfiguration of groups of sensor nodes at scale. RMD is a cross-layer solution, utilizing MAC layer information about neighborhood and packet losses. Moreover, RMD controls the MAC protocol in order to reduce idle listening to a minimum. Compared to other transport protocols for WSN, RMD ensures data delivery to all recipients even under high error rates (up to 50%), while consuming 2-3 times less energy and maintaining a comparable delay. Due to the cross-layer design, the resources demanded by RMD are very low: 2.7 kB of FLASH memory for the code and $\approx$ 190 bytes RAM for internal data. The detailed description and evaluation of RMD can be found in Chapter 4.

Figure 3.5b shows the successful deployment of a service based on business rules. The target nodes are identified by their group ID. The detailed log of the dissemination session (including the average speed, number and types of errors, etc.) is stored on the gateway and can be retrieved in the web console for analysis purposes.

3.6.3 Application Trials

To test our system in a realistic environment, we conducted two application trials at a BP petrochemical plant in UK. In both trials we equipped 20 chemical containers with wireless sensor nodes and distributed them in three different locations. A gateway was installed at each location, in order to connect the WSN to the WLAN and further to the back-end server. Each trial lasted about four weeks and consisted of two phases: the supervised phase (installation and manual testing) and the unattended phase (normal operation on-site). The goals of the first trial were to check if the WSN and the back-end system react correctly to real situations, and to measure the scalability of the overall solution. The second trial focused on improving the stability and scalability of the back-end system, and on prolonging the battery lifetime of the sensor nodes.

Both application trials confirmed the feasibility of our solution. During the second trial, for example, the system handled successfully a total number of 162294 messages from the sensor nodes, with an average message load of ≈ 7 messages/minute. The last day of the trial was allocated for testing under stress conditions, by placing all the 20 containers in the same location and generating continuously alarms. As a result, the average message load increased to 212 messages/minute, with a peak rate of 225 messages/minute. Even under these conditions, the WSN continued to operate reliably. However, the stress test revealed a scalability limitation of the UPnP eventing. The UPnP GENA uses TCP connections for eventing on subscriptions, which leads to a complete TCP connection setup on each event. Unless the client implements HTTP/1.1 pipelining and reuses callback sockets for multiple services, there is no way of reducing the overhead of the UPnP eventing. In the following, we briefly list other problems encountered during the application trials.

Firstly, the UPnP implementation used in the application trials (Cyberlink) is intended to serve a few services and therefore scales poorly for productive usage, by overwhelming the gateway with excessive multi-threading. Secondly, the real deployment conditions, such as the weather conditions, can adversely affect the performance of the system. For example, during the application trials, the 802.11 network exhibited high packet loss at times due to rather high humidity. This affected the UDP traffic of the UPnP discovery operation and the DHCP-based dynamic addressing for the routers. Finally, when dealing with safety-critical scenarios, the costs of intrinsically safe equipment adds a high factor concerning packaging and quality control. Even if the WSN technology is ubiquitous and cheap, these costs remain rather constant, since strict guidelines have to be fulfilled, such as the directive 94/9/EC (Equipment intended for use

in potentially explosive atmospheres – ATEX).

3.7 Conclusions

In this chapter, we addressed the integration of ubiquitous technologies into decentralized enterprise environments. The ultimate goal is to delegate well-defined parts of the business logic to the low-cost embedded devices, and thus reduce the process execution costs and improve the response time in safety-critical situations. We presented a layered service-oriented solution that accommodates three different sensor platforms and exposes their functionality seamlessly through the UPnP standard to the business process based on SAP enterprise software. Such a service-oriented system can be designed top-down, with business services of coarse granularity broken down into lower-level system services. Thereby, the communication between the business process experts and the technical experts is simplified and allows for a separation of concerns.

The practical tests and application trials confirm the feasibility of our solution. Multi-platform integration provides versatile functionality to the user, in a uniform way. The local, collaborative execution of tasks within the WSN reduces the load on the back-end and improves the overall responsiveness.

Our experiences also pointed out several directions for future work. Firstly, energy-efficiency remains a major concern for the battery-powered sensor nodes. This is why low duty cycle operation, lightweight execution models (such as business rules) and cross-layer solutions (such as RMD) should be favored. Secondly, increasing the level of context awareness, local intelligence and cognition of the deployed WSNs represents a strong point to foster the technology acceptance in industrial environments. These aspects will be studied in detail in the following chapters. Finally, the integration with service-oriented back-end frameworks involves a serious overhead that can result into scalability problems, as we encountered during our application trials. Potential alternative solutions to UPnP, such as DPWS (Devices Profile for Web Services) [69], are still to be explored.

Chapter 4

Reliable Multicast Data and Code Dissemination

This chapter addresses the problem of reliable data and code dissemination in WSNs. Starting from practical experiments with reliable data delivery in WSNs, we design, evaluate and implement an energy-efficient, reliable multicast dissemination solution. Furthermore, we analyze the performance of the entire network stack (data link, topology control and reliable dissemination) and study the impact of the deployment properties, as well as of the irregularities of the real wireless environment. This chapter combines the following three publications: "Experiments with Reliable Data Delivery in Wireless Sensor Networks" in Intelligent Sensors, Sensor Networks and Information Processing Conference (ISSNIP), December 2005 [122], "RMD: Reliable Multicast Data Dissemination within Groups of Collaborating Objects" in Local Computer Networks (LCN), November 2006 [123], which are joint work with P. Havinga, and "A Simulation Framework for Evaluating Complete Reprogramming Solutions in Wireless Sensor Networks" in International Symposium on Wireless Pervasive Computing (ISWPC), May 2008 [92], which is joint work with M. Horsman, P. Jansen and P. Havinga.

4.1 Introduction

In contrast to the Internet world, where TCP is the de-facto standard protocol, in WSNs there is no commonly-accepted meaning for *reliable transport*. The term is overused to encompass several distinct aspects: data collection, event reporting, data and code dissemination, network reprogramming, etc. Therefore, to avoid any confusion, we state from the beginning that this chapter addresses the problem of reliable data and code dissemination from a central point to groups of static sensor nodes.

Reliable data and code dissemination delivers a critical service in any practical application of WSNs: the dynamic reconfiguration of the network functionality. Referring back to the concrete use cases from Chapter 3, we can identify the following major requirements:

1. *Correct data delivery.* Despite the well-known unreliability of the wireless medium, the reliable dissemination solution must deliver correctly all the data to every intended recipient.
2. *Indication on failure.* There are situations when the data delivery is not possible, for example when the network becomes partitioned. Nevertheless, in those situations it is important for the application running on the back-end to have an indication of where the failure occurred, at least in terms of network topology if no geographic location is available.
3. *Flexibility.* Complex applications make use of heterogeneous sensor nodes, specialized on specific tasks. Consequently, targeting groups of nodes instead of the whole network adds to the flexibility of the overall system, as the user can selectively reconfigure the WSN.
4. *Energy efficiency.* The communication overhead of a reliable dissemination protocol is significant, especially in presence of high error rates. It is therefore essential to preserve energy efficiency and, more specifically, to minimize the number of transmissions, receptions, as well as the time spent with idle listening.
5. *Low latency.* Especially in the case of reprogramming the WSN, large code images may be sent into the network. The reliable dissemination protocol is expected to deliver all the information as fast as possible, so that the time the network is not operational stays at a minimum. Together with requirements 1 and 4 (delivery correctness and energy efficiency), latency creates a triangle of tradeoffs for any reliable dissemination protocol.

To answer the problems associated with these requirements, we propose RMD, a reliable multicast data and code dissemination solution. RMD makes two important design choices: (1) it relies on a tree structure, which allows both local and end-to-end error control and (2) it interacts with the MAC layer by using MAC acknowledgments (ACKs) and by applying a selective listening scheme to reduce idle listening time.

The remaining of this chapter follows our general methodology. In Section 4.2 we overview the relevant reliable transport solutions and we position our contribution with respect to related work. In Section 4.3 we study experimentally three prototype solutions, in order to better understand the real challenges in a WSN testbed. The core sections of this chapter, Section 4.4, 4.5 and 4.6, present the detailed design and analysis of RMD, including the performance comparison to PSFQ, a representative reliable transport protocol for WSNs. In Section 4.7 we extend the performance evaluation to the complete network stack and analyze the impact of different network properties to the overall system. Section 4.8 briefly overviews the implementation details on the prototype sensor node platform. Finally, Section 4.9 formulates the conclusions.

4.2 Related Work

The generic topic of reliable transport protocols accounts for an extensive research literature covering both wired and wireless networks. In the following, we focus on two specific areas that relate to our work: general-purpose multicast protocols and reliable transport solutions for WSNs.

Concerning the first topic, a comprehensive survey and taxonomy is given by Obraczka et. al [136]. The authors identify two major classes of applications that have motivated a whole suite of multicast protocols: multipoint interactive applications and data dissemination applications. Another interesting taxonomy [112], extending the study of Pingali et. al [140], describes four generic approaches for reliable multicasting: sender-initiated, receiver-initiated, tree-based and ring-based. It is shown that *tree-based protocols* constitute the most scalable class of all reliable multicast protocols and also the best choice in terms of processing and memory requirements. This result is of particular interest for the WSN community, and represents one of the main arguments of our approach.

The problem of reliable transport in WSNs has received increasing attention in recent years. Without being exhaustive, Table 4.1 overviews several well-known protocols. We distinguish three classes, according to their general

purpose:

1. Data collection and event reporting from sources to sinks
2. Complete network reprogramming solutions
3. Generic data and code dissemination

Table 4.1 also compares the main properties of reliable transport protocols in WSNs. Data type is closely related to the general purpose, for example data collection involves small, periodic messages, while reprogramming occurs occasionally, but requires large code images to be transferred. The recovery and acknowledgment methods are essential for characterizing a reliable transport protocol. The traditional design choices are sender-initiated ACK and receiver-initiated NACK, but WSN protocols added new techniques to reduce energy consumption and network traffic. The scope and topology refer to the intended receiver set and the underlying network structure assumed by the transport layer. End-to-end control guarantees to the source that the data was correctly transferred to the destination. Cross-layer is a WSN design choice that can improve energy consumption and latency, by using important information already available at lower layers. Finally, the last column of Table 4.1 gives the sensor node platform on which the specific protocol is implemented.

In the following, we briefly review the main characteristics of each protocol, as grouped by the three classes previously mentioned.

Data collection and event reporting from sources to sinks. The first class of protocols targets data collection and event reporting from sources to sinks. RMST [156] is a well-known example that works in conjunction with Directed Diffusion [95] to ensure data delivery and packet fragmentation and reassembly. The goal of RMST is to eventually deliver fragmented data to all the subscribing sinks, which are responsible for loss detection.

ReInForM [70] and MDR [182] protocols do not use an explicit acknowledgment scheme, but instead send multiple copies of the same packet over multiple paths from sources to sink. MDR reduces the traffic volume by splitting the data packets into subpackets and adding correction code redundant information. This allows to rebuild the original data packets at the destination even if not all the subpackets are received.

ESRT [152] and CODA [168] concentrate explicitly on controlling the congestion of burst data reported in case of events. ESRT regulates the reporting rate of source nodes so as to keep both the event detection reliability and the congestion within an optimal operation region. The control is centralized at the sink node, which is assumed to be powerful enough to reach all sources by

broadcast. CODA leverages this condition by incorporating three mechanisms for congestion detection and avoidance: receiver-based congestion detection, open-loop hop-by-hop backpressure and closed-loop multi-source regulation.

Complete network reprogramming solutions. The class of over-the-air reprogramming solutions includes the protocols designed specifically for transferring new program code images in the WSN. There are two basic approaches: entire code delivery and difference-based update. MOAP [157], Deluge [94] and MNP [107] follow the first approach and all use an *epidemic* message propagation. In MOAP, a node has to obtain the entire code image before becoming a source for its own neighborhood, while Deluge and MNP divide the code image into *pages* or *segments* that are transferred in a pipeline fashion. Compared to Deluge, MNP saves energy by avoiding idle listening, but has a higher latency.

Infuse [106] targets also entire code delivery, but, unlike MOAP, Deluge and MNP, relies on a TDMA scheme for collision-free communication and includes explicit recovery based on sliding-window protocols (Go-Back-N and selective retransmission). Additionally, Infuse introduces a *preferred predecessor* relation for further reducing the idle listening time. Sprinkler [133] is a protocol for reprogramming extremely large networks, developed in the ExScal project [11]. Similarly to Infuse, Sprinkler relies on TDMA communication. The important difference is that Sprinkler constructs a connected dominating set (CDS) of nodes, which are selected as senders in order to reduce energy consumption. The recovery phase is receiver-initiated and distributes the load for performing repairs among parent nodes.

The second approach, taken by Trickle [115] and ECD [146] [1] , propagates and maintains code updates instead of whole images, thus reducing the traffic volume. Trickle uses a "polite gossip" policy, where nodes broadcast periodically code updates (based on version numbers) to their direct neighbors. The algorithm is integrated with Maté [113], a tiny virtual machine for WSNs. ECD builds the code updates in a *diff*-like manner, using a custom set of binary opcodes to describe the edit scripts efficiently. Compared to Trickle, ECD can achieve complete node retasking, including the operating system, but focuses on generating the patches rather than transporting the data in the network.

Generic data and code dissemination. From the class of generic dissemination protocols, uIP [71] represents a minimal implementation of the TCP/IP stack for sensor nodes. On the one hand, uIP has the advantage of following a standard paradigm, but on the other hand, it requires an IP-like routing

[1] In the absence of an acronym given by the authors, we use ECD for the "Efficient Code Distribution" proposed by Reijers and Langendoen [146].

substrate and can suffer from limited performance and energy inefficiency in large-scale, multihop WSNs.

PSFQ [167] is one of the first reliable transport protocols for WSNs and achieves hop-by-hop error recovery through receiver-initiated NACKs. The key idea is to slowly inject messages (the *pump* operation) into the network, while quickly performing repairs from direct neighbors (the *fetch* operation). The ratio between the pump and the fetch timers provides a number of possible retransmissions of the missed packets.

GARUDA [139] disseminates queries from sink to sources and constructs a loss recovery infrastructure (the *core*) for quickly repairing the missed packets. The *core* is built based on hop count information and approximates a minimum connected dominating set (MCDS). Since the recovery scheme is receiver-initiated, GARUDA increases the chance of delivering the first packet by sending a Wait-for-First-Packet (WFP) pulse. McCORD [93] is another core-based reliable transport protocol, designed for bulk data dissemination. The usage of the core infrastructure for repairing missing packets is similar to GARUDA. The construction of the core takes link quality into account, thus selecting only nodes connected through stable links. Additionally, McCORD uses *multi-channel communication* to reduce object delivery latency.

The design choice of Typhoon [117] is to reliably deliver large objects to all the nodes with *minimal idle listening time*. Unlike most other protocols, Typhoon sends data packets through unicast and implements a stop-and-wait ARQ protocol. Similarly to McCORD, Typhoon introduces channel switching to accelerate the propagation of data through the network.

Our solution, RMD [123], relies on a multicast tree to distribute data or code to groups of nodes. RMD exploits the tree structure and the information from the MAC layer to keep the responsibility of repairing missed packets within the local neighborhood, while still ensuring end-to-end error control.

Concluding remarks. The analysis of the previous protocols leads to the following concluding remarks:

1. Protocols are typically specialized, due to the extreme constraints of WSNs. Therefore, *there is no one size fits all solution at the current moment.*
2. Very few protocols ensure end-to-end error control, as the multihop communication is expensive in WSNs. However, *without any form of end-to-end control, there is no way the back-end application can guarantee to the user that the data was transferred correctly to all intended recipients.*
3. Being an effective loss advertisement mechanism, receiver-initiated NACKs are the usual choice. Nevertheless, without any explicit ACK, a sender

node cannot know when to safely advance to the next window of packets and free its memory. In other words, *the liveness property of receiver-initiated, NACK-based protocols is affected* (see [112] for the detailed proofs).

4. The interaction with lower layers (physical, data link and routing) is generally ignored, although they can render useful information and functionality, such as radio link quality, neighborhood status, ACKs and automatic retransmissions. Therefore, *cross-layer interactions leverage significantly the overhead of the reliable transport protocol, while also increasing the chance of successful data delivery.*

Table 4.1: Protocols

Protocol	Purpose	Data type	Recovery	ACK scheme	Scope	Topology	End-to-end	Cross-layer	Platform
RMST [156]	data collection	small, periodic	receiver initiated	NACK	multicast	mesh, Directed Diffusion	no	yes	–
ReInForM [70]	data collection	small, periodic	multipath	–	unicast	mesh	no	no	–
MDR [182]	data collection	small, periodic	multipath, correction codes	–	unicast	mesh	no	no	EYES [8]
ESRT [152]	event reporting	small, burst	congestion control	–	unicast	mesh	yes	no	–
CODA [168]	event reporting	small, burst	congestion control	–	multicast	mesh	yes	no	Mica mote [7]
MOAP [157]	reprogram	large, seldom	epidemic	NACK	broadcast	mesh	no	no	Mica mote [7]
Deluge [94]	reprogram	large, seldom	epidemic	NACK	broadcast	mesh	no	no	Mica mote [7]
MNP [107]	reprogram	large, seldom	epidemic	NACK	broadcast	mesh	no	no	Mica mote [7]
Infuse [106]	reprogram	large, seldom	sender initiated	implicit ACK	broadcast	mesh	no	no	Mica mote [7]
Sprinkler [133]	reprogram	large, seldom	receiver initiated	NACK	broadcast	CDS	no	no	XSM [11]

Continued on next page

Table 4.1

Continued from previous page

Protocol	Purpose	Data type	Recovery	ACK scheme	Scope	Topology	End-to-end	Cross-layer	Platform
Trickle [115]	reprogram	medium, seldom	gossip	–	broadcast	mesh	no	no	Maté [113]
ECD [146]	reprogram	small, seldom	diff-based	–	broadcast	mesh	no	no	EYES [8]
uIP TCP/IP [71]	generic	any, any	sender initiated	ACK	unicast	mesh, IP	yes	no	Tmote [18]
PSFQ [167]	generic, dissemination	large, seldom	receiver initiated	NACK	multicast	mesh	no	no	Mica mote [7]
GARUDA [139]	query dissemination	small, periodic	receiver initiated	NACK, WFP pulse	multicast	core-based (MCDS)	no	no	–
McCORD [93]	generic, dissemination	large, seldom	receiver initiated	NACK	broadcast	core-based (MCDS)	no	no	Mica and TelosB mote [7]
Typhoon [117]	generic, dissemination	large, seldom	sender initiated	ACK	broadcast	mesh	no	no	Mica mote [7]
RMD [123]	generic, dissemination	any, seldom	sender initiated	ACK, MAC ACK	multicast	tree-based	yes	yes	Ambient μNode [1]

4.3 Initial Experiments

This section presents initial practical tests with reliable data delivery between a source and a destination node over a variable number of hops. We study three reliable transport methods bearing increasing complexity:

- Protocol 1: End-to-end acknowledgment for every packet (stop-and-wait ARQ).
- Protocol 2: End-to-end, window-based acknowledgment (TCP approach).
- Protocol 3: Local error detection and repairs, plus end-to-end, window-based acknowledgment.

Recalling the reliability requirements of industrial applications, such as the ones presented in Chapter 3, we consider compulsory to have an end-to-end form of acknowledgment. Otherwise, the back-end system would have no means to guarantee, for example, that a service deployment completed correctly.

In order to have a basis for our comparisons, we first implement the simple stop-and-wait ARQ, which requires explicit acknowledgment of each packet.

The second solution we explore relies on window-based selective ACK/NACK. More specifically, the sender transmits the packets within the current window and then waits for either a complete ACK or a selective NACK indicating the missing packets. The selective NACK packet contains the binary status of the window at the receiver side; for instance `11011` means that the third packet was not correctly received.

So far, the intermediate nodes just route the data packets between source and destination, while error control is done exclusively end-to-end. The third protocol works in conjunction with the MAC layer, which provides local ACKs at low energy costs. Every intermediate node on the path from source to destination maintains a local copy of the current window, from which it periodically sends one packet to the next hop neighbor. Each packet is acknowledged at the MAC level or otherwise retransmitted. The entire window is acknowledged by a selective ACK/NACK as in the previous case.

4.3.1 Experimental Setting

For our experiments, we use up to 6 nodes placed linearly in an indoor (office) environment. Each experiment consists in transmitting a message of 400 bytes from the PC, through the serial port, to a gateway sensor node and further to the intended destination node. The transmission is fully streamed, so that the

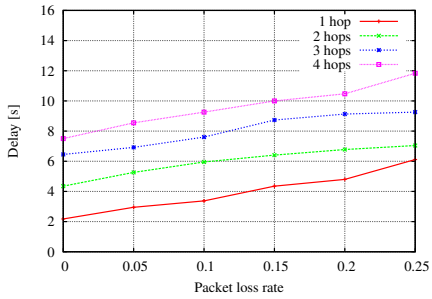
Protocol	**Node type**	**Code** (FLASH)	**Data** (RAM)
Protocol 1	Intermediary	716	8
	Endpoint	880	20
Protocol 2	Intermediary	716	8
	Endpoint	1034	34
Protocol 3	Intermediary	1038	282
	Endpoint	1034	34

Table 4.2: Code and data memory footprints for the three reliable transport methods (figures given in bytes). RAM size includes protocol buffers. The packet size is set at 32 bytes.

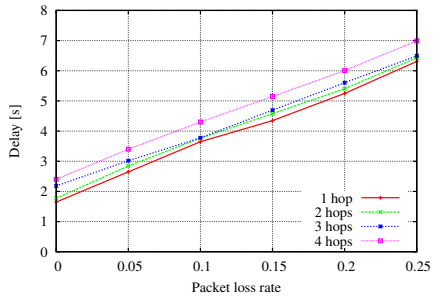
gateway node acts only as a bridge. The communication on the serial link is done reliably using stop-and-wait ARQ. The source starts by sending an initial announcement packet, used for setting up the session, the sequence numbers and the routing path, next it progressively sends the message, divided in fixed size packets, and waits for acknowledgments according to the current protocol scheme. Table 4.2 summarizes implementation details such as code and data memory footprints both for endpoints and intermediary nodes.

We analyze two performance metrics: *latency* and *goodput*. Latency is expressed as the delay recorded for correctly delivering one data packet. Goodput is computed as the amount of useful information successfully transmitted, scaled to the total communicated data (including headers, acknowledgments, retransmissions, etc.). This offers a relevant indication on the overhead and the energy efficiency of the protocol.

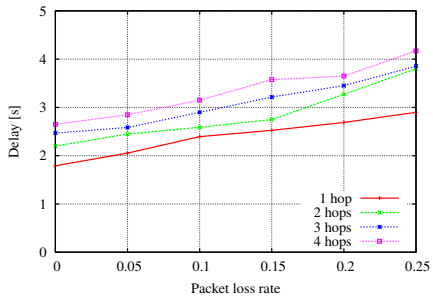
Both the latency and goodput are influenced by two parameters: *hop distance* and *packet loss rate*. The hop distance depends on the particular physical deployment of the nodes and is known before starting the transmission. For protocols 1 and 2, the packet loss rate is determined at endpoints, whereas for protocol 3, all the intermediary nodes count the number of local errors and, after message completion, they report back to the PC. On the duration of each experiment the nodes are static, as we do not aim to study the impact of mobility. However, we vary the distance between nodes in different experiments, so that we obtain various packet loss rates. This procedure requires a large number of experiments in order to have a sufficiently large pool of data for analysis. Depending on the actual packet loss rate observed, each point in Figures 4.1 and 4.2 represents an average over 5 to 10 experiments.



(a) Protocol 1



(b) Protocol 2



(c) Protocol 3

Figure 4.1: Average latency of the three protocols as a function of the packet loss rate and the number of hops.

4.3.2 Results

The graphics from Figure 4.1 depict the average latency as a function of the packet loss rate, for hop distances between 1 and 4. We can see that the protocol 1 exhibits substantial overhead and its performance decreases drastically for multi-hop settings. This solution should be applied only when there are strict limitations on the size and complexity of the code that runs on the nodes.

Protocol 2 performs better, especially for low error rates (below 10%), as it requires fewer acknowledgments and permits the receiver to request retransmissions only for the lost packets from the current window. The major implementation difficulty consists in finding the proper values for the window size and the timeout timers. A larger window means fewer ACK/NACK packets, but also longer delays in case of errors. Both Figures 4.1 (b) and (c) represent results

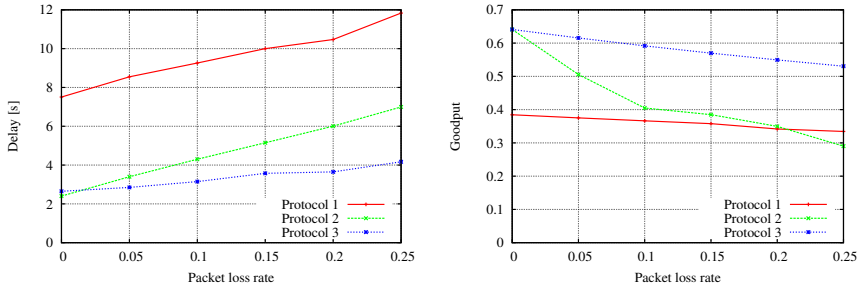


Figure 4.2: Comparison of (a) average latency and (b) goodput for 4 hops.

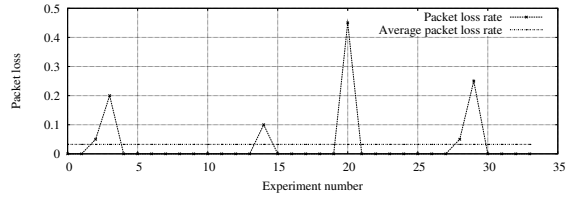
considering a window size of 5 packets, which experimentally proved to be an acceptable tradeoff.

Figure 4.1 (c) shows that local error detection and retransmissions attenuate the effect of communication errors. However, this comes at the cost of reserving memory for caching the current window on each intermediary node. For low error rates, we notice additional latency compared to protocol 2. This is explained by the fact that the nodes have to store the incoming packets and schedule them for sending, therefore their time slot is occasionally ended before the actual transmission.

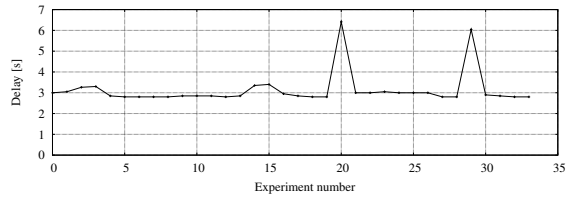
Figure 4.2 provides a comparative view on the performance (average latency) and efficiency (goodput) for 4 hops. It is interesting to notice that protocol 3 is not always the best choice with respect to latency. Indeed, for error rates less than 5%, the second protocol exhibits almost similar performance. Nevertheless, in terms of goodput, performing local repairs reduces the number of ACK/NACK packets at the transport layer and therefore protocol 3 proves to be the most efficient.

We perform a final test using protocol 3, by running 35 continuous experiments over 5 hops, transferring in total approximately 14kB of data. The results in Figure 4.3 show that communication errors generally determine burst losses, which in turn affect the overall transmission speed. The average recorded packet loss is 3.3%. [2]

[2]This value depends of course on many factors, such as the experiment setting, radio chip, etc. However, later experiments by Willig and Mitschke [179] confirm our results. For a total of 9000 packets transmitted, they recorded packet loss rates from 0.03% to 9.8%, with an average of 3.8%.



(a) Packet loss



(b) Average latency

Figure 4.3: Sending a total of 14kB over 5 hops using protocol 3: (a) packet loss and (b) latency.

4.3.3 Conclusions

This section presented our early experiments with prototyping reliable transport in WSNs. As a feasibility confirmation, we saw that reliable transport solutions can be implemented in 1kB memory footprint on sensor nodes. As listed in Table 4.2, local error control (protocol 3) allocates more RAM for the local window cache than the exclusive end-to-end methods. However, the general performance of protocol 3 is much better under high error rates.

With respect to the properties of WSN communication, our practical experiments indicate that *errors usually occur in burst and thus determine the loss of one or more packets*. Possible reasons include losing connectivity to a neighbor from the routing path, radio interferences and hardware problems.

Therefore, the main recommendations derived from our experiments are:

1. Traditional end-to-end error control should not be applied directly, but combined with local error detection and recovery.
2. Errors usually occur in burst and determine the loss of one or more packets. Therefore, a transport protocol should mainly handle packet losses, and rely on the MAC layer to deal with bit errors.

3. A cross-layer design, where the transport layer interacts with routing and MAC, can improve the energy and throughput performance.

Taking into account these results, we propose in the following RMD, an energy-efficient, tree-based reliable multicast dissemination protocol for WSNs.

4.4 RMD Protocol Description

In this section, we first discuss the considerations that underpin the design of RMD, then describe the protocol operation and discuss the most important parameters and assumptions.

4.4.1 Design Considerations

Tree-based Reliable Multicast

As already mentioned in Section 4.2, tree-based reliable multicast protocols (TRMP) [112] are shown to exhibit good scalability properties by delegating responsibility to local groups (i.e., a node and its children in the tree). An additional advantage of TRMP is that they can contribute to energy efficiency, by saving a considerable amount of network traffic through local and aggregate ACKs. The local ACKs trigger local retransmissions from a parent node to its children. The aggregate ACKs indicate correct overall reception to the source: a parent node will send an aggregate ACK only after it receives aggregate ACKs from all its children. Hence, the ACK-implosion problem is avoided and the source does not need to know the whole receiver set [112]. The local behaviour of TRMP makes them suitable for the case of dense WSNs. Although WSNs usually rely on a mesh model, a tree or cluster overlay organisation is often required for a proper, distributed operation. The reliable multicast protocol can thus directly use the available overlay structure.

There is an extensive range of tree-based multicast routing protocols proposed in the ad-hoc and sensor network literature. A comprehensive overview of the most important approaches can be found in [132]. The construction and maintenance of the multicast tree is therefore beyond the scope of RMD. Nevertheless, in order to thoroughly evaluate the performance of RMD, we use FixTree (see Section 4.7), a source-initiated tree construction protocol. FixTree is a basic protocol that builds the spanning tree by selecting the most reliable links and releases the tree at the end of the dissemination session. Since FixTree

does not attempt to maintain or repair the tree, it acts as a baseline for more advanced protocols that would further increase the data delivery performance.

Cross-layer Design

According to the recommendations derived from our experiments in Section 4.3 results, we design the tree-based reliable dissemination protocol to work jointly with the MAC layer. We assume that the MAC layer provides the following functionality and information (see LMAC [89] for an example):

- Neighbor information, from which a node can derive the identity of its parent and children in the tree, as well as a change in its neighborhood (a broken link or a new neighbor).
- Local (one-hop) acknowledgments with minimal energy expenditure.
- Interaction points (callbacks) for signaling local ACKs and changes in neighborhood to the dissemination layer.

The RMD protocol is responsible for the following operations:

- Fragment the message into fixed size windows of packets (the packets are identified by sequences numbers).
- Ensure end-to-end delivery through window ACKs.
- Control the MAC layer for the listening phase, in order to save energy (see the local multicast primitive defined in Section 4.5.4).

It follows that we distinguish between local ACKs (one hop, MAC-based, one for each packet), and window (or aggregate) ACKs (end-to-end, handled by the dissemination layer, one for each window). The reason is that MAC-based ACKs and retransmissions consume less time and energy than transport layer ACKs. With respect to the requirements defined in Section 4.1, local ACKs and cross-layer listening form the mechanisms to ensure energy efficiency, while end-to-end aggregated ACKs guarantee the delivery and allow the source node to safely release the data from the memory.

4.4.2 Protocol Operation

RMD assumes that the initiator of the dissemination session, also referred to as *source* node, is the root of the multicast tree. The tree comprises the multicast group members (the intended receivers) and the additional forwarders needed for

propagating the data. A node can therefore be *active* or *passive* for the current dissemination session. According to the position in the tree, we distinguish between *source*, *intermediate* and *leaf* nodes (see also Figure 4.4(a)). Due to the tree structure, the *parent-children* relation between nodes is important for protocol operation. We assume that the source and intermediate nodes have on average B children, i.e. the average tree degree is B.

Algorithms 1 and 2 describe the RMD operation. The protocol comprises two phases: initialization and running.

Algorithm 1: RMD initialization

```
Data:
  p - parent node ;
  K - list of children ;
  M - list of children that sent MAC ACKs, M ⊂ K, initially M = φ ;
  A - list of children that sent window ACKs, A ⊂ K, initially A = φ ;
  r - number of retransmissions, at most r_max, initially r = 0 ;
switch packet received do
    case NEW_MESSAGE
        init protocol variables, sequence number ;
        if K = φ then                                          // leaf node
            send ACK to p ;
        else
            send NEW_MESSAGE to K ;
        end
    case ACK from v
        A = A ∪ {v} ;                                      // mark child ACK
        if A = K then                                 // all children sent ACKs
            send ACK to p ;
            A = φ ;
        end
end
```

In the *initialization phase*, a `NEW_MESSAGE` packet traverses the tree, announcing the start of the dissemination session. Using this packet, the height of the tree is computed. The initialization phase ends when the root receives explicit aggregated ACKs from all its children.

In the *running phase*, the source starts sending `DATA` packets from the current window, which are further pipelined down the tree by the intermediate nodes. The sending occurs periodically at the constant timer τ_1. At all levels in the tree, each packet is resent until all the direct children acknowledge it through MAC ACKs (see Figure 4.4(b)). The MAC ACKs are signaled through the callbacks defined by the dissemination layer. The leaves of the tree indicate the correct

reception of an entire window through window ACKs. Further, the window ACKs travel back to the root by being aggregated at every intermediate node (see Figure 4.4(a)). Upon receiving the window ACKs from all direct children, the root node advances to the next window of packets.

Due to the unreliable wireless channel, a certain number of errors may occur during RMD operation:

- *Packet loss.* Packets may be lost or corrupted when transmitted from parent to children. As a consequence, the parent node does not receive MAC ACKs from all its children and the current packet is resent at the next time step τ_1. After the maximum number of retransmissions r_{max} is exceeded, the non-responsive children are declared lost and reported upstream to the root. The maximum number of retransmissions is a parameter to be determined experimentally; for an example see Section 4.7.5.
- *Link breakdown.* The link between a node and its parent may become unavailable. We assume that this change in the neighborhood table is signaled by the MAC layer through a `LINK_DOWN` callback. The child node has to register to another parent, if available, by sending a `RECOVERY_REQ` message. The exact recovery procedure and the delay induced are dependent on the underlying tree maintenance protocol and therefore difficult to account for at the transport layer. It is however reasonable to assume that the node will require from the new parent a partial (and, in the worst case, an entire) update of the current window of packets.
- *Network partition.* In the case when no alternative parent is available, the node and its sub-tree remain isolated from the network. The dissemination will continue without them (indication of failure is sent to the root node, though). If later on the network connection recovers, any node that received correctly all the data can act as a dissemination source and initiate RMD to update the disconnected sub-tree.
- *Hardware error.* Nodes may reset due to imperfect battery contacts, harsh environments, watchdog behavior, etc. RMD ensures correct delivery even in presence of such errors. Nodes that are intended receivers (i.e. members of the multicast group) store every correctly received data packet in non-volatile memory. Nodes that are just data forwarders cache only the current window, from which they perform local repairs. After reboot, a node experiencing a hardware error resumes its state from the non-volatile memory and, upon hearing the first packet from the parent, requests recovery through the `RECOVERY_REQ` message.

Algorithm 2: RMD operation

Data:
p - parent node ;
K - list of children ;
M - list of children that sent MAC ACKs, $M \subset K$, initially $M = \phi$;
A - list of children that sent window ACKs, $A \subset K$, initially $A = \phi$;
r - number of retransmissions, at most r_{max}, initially $r = 0$;

switch *event* **do**
  case τ_1 // sending timer
    if $r > r_{max}$ **then** // some children are lost
      inform children lost $(K - M)$;
      advance to next packet in window ;
      $K = M$, $r = 0$, $M = \phi$;
    else
      send current packet in window to K ;
      increment r ;
    end
  case *MAC ACK from* v // MAC ACK callback
    $M = M \cup \{v\}$; // mark child MAC ACK
    if $M = K$ **then** // all children sent MAC ACKs
      advance to next packet in window;
      $r = 0$, $M = \phi$;
    end
  case *LINK_DOWN to* v // link down callback
    if $v = p$ **then** // parent lost
      register to alternative parent p' ;
      send RECOVERY_REQ to p' ;
    else if $v \in K$ **then** // child lost
      inform children lost (v) ;
      $K = K - \{v\}$;
    end
  case *DATA from* p // data packet received
    check sequence number and store packet ;
    if $K = \phi$ *and window complete* **then** // leaf node received all window
      send ACK to p ;
    end
  case *ACK from* v // window ACK received
    $A = A \cup \{v\}$; // mark child ACK
    if $A = K$ **then** // all children sent ACKs
      send ACK to p;
      advance to next window ;
      $A = \phi$;
    end
  case *RECOVERY_REQ from* v // recovery request received
    $K = K \cup \{v\}$; // add new child
    restart to send window from the beginning ;
end

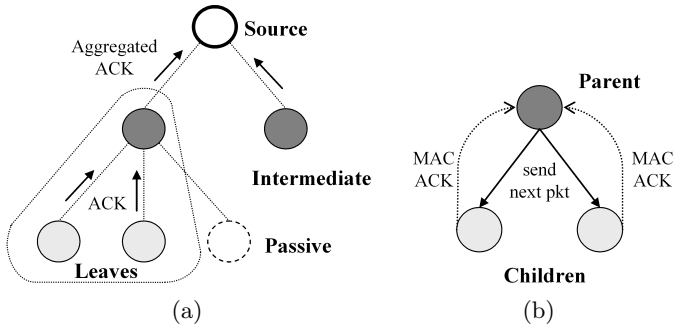


Figure 4.4: RMD protocol operation: (a) Tree structure and window ACK propagation, (b) MAC ACKs and retransmissions.

4.4.3 Assumptions

Throughout the rest of this chapter, we make a series of assumptions that simplify the presentation and analysis of the RMD protocol.

Firstly, we assume that packet losses occur with probability p_1 and packet loss events at all receivers are mutually independent. Consistent to the error model used in previous studies [140, 112], possible losses of window ACKs are ignored, in other words, window ACKs are never lost. [3]

Secondly, with respect to MAC cross-layer interaction, we consider the MAC ACKs to consume negligible time and energy compared to data packets and window ACKs. Furthermore, we assume that packet losses are reliably detected through MAC ACKs. As a consequence, we do not use selective ACK/NACK for window confirmation as in Section 4.3, but simple window ACK packets.

Finally, we denote by *faults* the situations of link breakdown or hardware error described above. For simplifying the analysis, we assume that a fault determines the retransmission of the entire current window. The probability of a fault is referred to as p_2. The case of network partition will not be considered in the analysis, as it represents a re-instantiation of the protocol with a new source node.

[3] This model is used for simplifying the protocol analysis and is based on the fact that ACK packets are one order of magnitude smaller and fewer than data packets. If desired, this assumption can be relaxed by following the methodology from [162].

4.5 Protocol Analysis

We continue in this section with a theoretical analysis covering the RMD protocol correctness, the estimation of the energy consumption and latency, and the efficiency in reducing the idle listening time.

4.5.1 Protocol Correctness

In order to show that RMD is correct, we have to prove that it delivers *all* the data to *every* intended recipient within a finite time [112]. We use two important results of previous work: (1) the unicast ARQ protocol is correct [46] and (2) the generic tree-based reliable multicast protocol (TRMP) is correct [112]. We must extend these results to show that the correctness property holds also in our hybrid error control scheme (both local, MAC-based and end-to-end, window-based). With respect to the protocol operation presented in Section 4.4.2, we make the following assumptions:

(a) There is a non-zero probability that a packet is received successfully,

(b) There is a non-zero probability that no faults occur during the transfer of one window,

(c) All nodes have finite memory.

The proof proceeds by induction on h, the height of the multicast tree. For $h = 1$ we have one source node and B receivers. Assume the source starts sending a window of size w at time t_1. Let $t_{1,i}$ be the time the source sends packet i, where $1 \leq i \leq w$, and $t_{2,i}$ the time all the receivers have issued MAC ACKs, respectively. From the proof for unicast ARQ protocols [46] and assumption (a), it follows that $t_{1,i+1} > t_{2,i} > t_{1,i} \geq t_1$ and $t_{1,i+1} < \infty$. The source therefore waits for all receivers to acknowledge at MAC level before advancing to next packet in window. If a fault occurs, the source restarts to send the window. The receivers ignore the packets they already have. Now, let t_2 be the time all the receivers have sent window ACKs and t_3 the time the source advances to the next window. Using assumption (b) and the previous result, it follows that

$$t_3 > t_2 > t_{2,w} > t_{1,w} > t_{2,w-1} > ... > t_{1,i} \geq t_1 \tag{4.1}$$

and $t_3 < \infty$, which concludes the proof for this case.

We can proceed now to the induction step. Assuming the protocol is correct for $1 < k < h$, we must prove for $k = h$. Consider the subtree of height $h - 1$,

Symbol	Description
p_1	Packet loss probability (current packet is retransmitted)
p_2	Fault probability (current window is retransmitted)
w	Window size
B	Average number of receivers (or tree degree)
T_1^B	Number of transmissions due to packet losses (B receivers)
T_2^B	Number of transmissions due to faults (B receivers)
$T_{S,I,L}$	Number of transmissions for source, intermediate and leaf nodes
$R_{S,I,L}$	Number of receptions for source, intermediate and leaf nodes
ξ_T, ξ_R	Energy cost for transmitting/receiving a packet
$X_{S,I,L}$	Total energy costs for source, intermediate and leaf nodes
h	Tree height
τ_1	Sending timer
D	End-to-end delay for transmitting correctly a window

Table 4.3: List of notations

rooted at the source. The correctness property is true in this subtree, according to the inductive hypothesis. The remaining nodes, i.e. all the nodes of level h, have their parents in the subtree. It suffices to show the correctness for an arbitrary group consisting of one parent from the subtree and its B children of level h. This reduces to the case $h = 1$, proved by Eq 4.1, with the single difference that the parent node is not the original source. However, since the protocol works correctly in the subtree of height $h - 1$, the parent node is live and represents a valid source for its children. This concludes the proof.

4.5.2 Energy Consumption

In order to derive theoretically the energy consumption of RMD, we estimate the number of packets sent and received by a node during the transmission of one window. For convenience, Table 4.3 lists the relevant symbols that we will use throughout the analysis. As general notation, we use $E[X]$ to represent the expected value of variable X.

Let us assume the simple case of one sender and one receiver. If p_1 is the probability of a packet loss, then the expected number of transmissions required for a correct reception $E[T_1]$ is given by the geometric distribution:

$$E[T_1] = 1/(1 - p_1) \tag{4.2}$$

Now consider that the receiver can experience a fault with probability p_2

at time t_i corresponding to the reception of the i^{th} packet. [4] As a result, the sender has to resend the entire window, according to our assumption on faults from Section 4.4.3. We refer to this as a new *attempt*. The probability that a fault occurs during one attempt is $p_{2w} = 1 - (1 - p_2)^w$, where w is the window size. The expected number of attempts until a successful window transmission is:

$$E[N_2] = 1/(1 - p_{2w}) = 1/(1 - p_2)^w \tag{4.3}$$

The number of faults is therefore $E[N_2] - 1$. The number of faults can also be written as:

$$E[N_2] - 1 = p_2 E[T_2] \tag{4.4}$$

where $E[T_2]$ is the expected number of transmissions required for the correct reception in presence of faults.

From Equations 4.3 and 4.4, we have:

$$E[T_2] = \frac{1/(1 - p_2)^w - 1}{p_2} \tag{4.5}$$

Let us analyze now the case of multiple receivers, where we denote by B the average number of receivers (i.e., the average tree degree). According to our assumptions from Section 4.4.3, the errors at different receivers are independent events. We must compute the expected number of transmissions needed until all B receivers get the packet correctly, $E[T_1^B]$. In this case, Pingali et al. [140] showed that:

$$E[T_1^B] = \sum_{k=1}^{B} \binom{B}{k} (-1)^{k+1} \frac{1}{1 - p_1^k} \tag{4.6}$$

Equation 4.6 takes into account the fact that once a receiver has correctly received a packet, future reception errors for the same packet retransmissions are no longer relevant at that receiver. In contrast, faults can occur during any attempt of sending a window. Therefore, the probability of a fault during one attempt is $p_{2w}^B = 1 - (1 - p_2)^{wB}$. Accordingly, the number of faults and transmissions due to faults in the case of B receivers, $E[N_2^B]$ and $E[T_2^B]$, are:

$$E[N_2^B] = 1/(1 - p_2)^{wB} \tag{4.7}$$

[4] Although fault probability analysis is typically time-based, e.g. in Poisson processes, in this case we link faults to the flow of packets because we are eventually interested in the number of packets sent and received. An additional argument in favor of this simplification is that packets are transmitted at regular time intervals given by the τ_1 timer.

$$E[T_2^B] = \frac{1/(1-p_2)^{wB} - 1}{1-(1-p_2)^B} \tag{4.8}$$

We can compute now the energy costs at the source, intermediate and leaf nodes, for sending a complete window and receiving the corresponding ACKs. We note with $X_{S,I,L}$ the energy consumption, with $T_{S,I,L}$ the number of transmissions and with $R_{S,I,L}$ the number of receptions (see also Table 4.3).

Source node. For successfully sending a window, the source node sends $E[T_S] = E[T_2^B]\,E[T_1^B]$ and receives B ACKs from direct children, so $E[R_S] = B$. Consequently:

$$\begin{aligned} E[X_S] = &\quad \xi_T\,E[T_S] \;+\; \xi_R\,E[R_S] \\ E[X_S] = &\quad \xi_T\,E[T_2^B]\,E[T_1^B] \;+\; \xi_R\,B \end{aligned} \tag{4.9}$$

Leaf node. A leaf node sends one ACK per window and receives all the packets sent by the parent through local broadcasts, except its own packet losses. Therefore, $E[T_L] = 1$ and $E[R_L] = E[T_2^B]\,E[T_1^B]\,(1-p_1)$, leading to:

$$\begin{aligned} E[X_L] = &\quad \xi_T\,E[T_L] \;+\; \xi_R\,E[R_L] \\ E[X_L] = &\quad \xi_T \;+\; \xi_R\,E[T_2^B]\,E[T_1^B]\,(1-p_1) \end{aligned} \tag{4.10}$$

Intermediate node. An intermediate node carries out the double task of distributing packets and aggregating ACKs to and from its children. In addition, it is actively involved in the dissemination process, by maintaining the current window in the cache, in order to perform local repairs. The energy costs of a source node and a leaf node are thus combined, $E[T_I] = E[T_S] + E[T_L]$, $E[R_I] = E[R_S] + E[R_L]$:

$$\begin{aligned} E[X_I] = &\quad \xi_T\,E[T_I] \;+\; \xi_R\,E[R_I] \\ E[X_I] = &\quad \xi_T\,(E[T_2^B]\,E[T_1^B] + 1) + \xi_R\,(E[T_2^B]\,E[T_1^B]\,(1-p_1) + B) \end{aligned} \tag{4.11}$$

Dividing the results from Equation 4.9 - 4.11 by the window size w yields the average energy costs per packet, which will be used in the results from Section 4.6.1.

4.5.3 Latency

In order to examine the average latency, we derive an approximation of the transmission time required for the correct delivery of one window. We assume

that the transmission time is mainly affected by the delay on the longest branch of the tree. We compute therefore the expected delay $E[D]$ along the branch of length h, where h is the level of the tree. In the absence of errors, as the transmission process is pipelined, the time $D^\star$ needed to transmit a window of size w from source to leaf and to get back the aggregated ACK is:

$$D^\star = \tau_1(w + h - 1) + \tau_1 h = \tau_1(w + 2h - 1) \tag{4.12}$$

In presence of errors, any retransmission occurring along the pipeline increases the delay with one time unit τ_1. The source nodes sends on average $E[T_S]$ packets per window instead of w. Each of the $h - 1$ intermediate nodes along the branch adds on average $E[T_I] - w$ packets due to retransmissions. As mentioned in the assumptions from Section 4.4.3, possible losses of window ACKs are ignored. In conclusion, we have to replace the term w in Equation 4.12 with the total number of packets transmitted by the source and intermediate nodes. This leads to:

$$E[D] = \tau_1 \left(E[T_S] \; + \; (E[T_I] - w)(h - 1) \; + \; 2h - 1\right). \tag{4.13}$$

where $E[T_S]$ and $E[T_I]$ are defined in Equations 4.9 and 4.11. The final average latency per packet is given by the ratio $E[D]/w$.

4.5.4 Reducing the Idle Listening Time

The cross-layer design of RMD (see Section 4.4.1) allows to minimize the idle listening time, which is a major source of power consumption [183]. In this section, we show how cross-layer efficient listening can improve both latency and energy performance.

Typically, the MAC layer provides two types of local (one-hop) communication primitives: unicast and broadcast. Let us assume that a parent node has n neighbours, out of which B are children involved in the dissemination process. The total energy spent for sending and receiving a packet from parent to children is $X_{ucast} = B\xi_T + B\xi_R$ for unicast, and $X_{bcast} = \xi_T + n\xi_R$ for broadcast packets, respectively. If we assume $\xi_T \approx \xi_R$, it follows that broadcasts are a good choice only for dense multicast groups:

$$B \geq (n + 1)/2 \tag{4.14}$$

However, in the case of unicasts, the delay is B times higher ($D_{ucast} = BD_{bcast}$) and the sender performs B transmissions for each packet (unbalanced energy consumption).

Controlling the MAC layer during the listening phase can offer a better alternative. More specifically, RMD can instruct the MAC to listen only for those packets sent by the parent node and intended for the specific multicast group. In this way, unnecessary listening is avoided, and the delay and energy consumption are minimal:

$$D_{mcast} = D_{bcast}, \quad X_{mcast} = \xi_T + B\xi_R \tag{4.15}$$

4.6 Performance Evaluation

In this section, we examine the energy and latency performance of RMD. First, we analyze the theoretical results from Section 4.5, then we compare via simulation with a well known transport solution, Pump Slowly Fetch Quickly (PSFQ) [167]. Throughout the rest of this section, we consider the sending timer τ_1=1s and the energy for one transmission/reception $\xi_T \approx \xi_R$=0.2mJ, which corresponds to a 32 bytes packet communicated using the nRF905 transceiver [88, 20]. The energy spent at the MAC level for administrative overhead, e.g. for synchronization, and at the physical level, e.g. for powering on and off the transceiver, is not taken into account in this evaluation.

4.6.1 Theoretical Results

Figure 4.5 summarizes the theoretical results for energy consumption, when varying different parameters. Figure 4.5(a) shows the average energy costs for source, leaf and intermediate nodes, as a function of the packet loss rate p_1, for an average tree degree $B = 4$ and the fault rate p_2=1%. As expected, the intermediate nodes are more loaded than the source or leaf nodes, since they are involved in both downstream and upstream traffic, and perform local repairs. An analysis on the separate transmissions and receptions shows that the intermediate nodes spend comparable energy on both operations, whereas the source and the leaf nodes execute mostly one operation, according to their role. Figure 4.5(b) illustrates the exponential influence of the fault rate p_2 on the energy consumption of the intermediate nodes. Figure 4.5(c) reflects the effect of the average tree degree B, under various packet loss rates. We note that the local recovery and ACKs aggregation mechanisms ensure a linear performance degradation, even under high packet loss rates. Figure 4.5(d) shows the variation of energy costs on intermediate nodes when varying the window size w. We see that increasing the window size has a positive effect up to a certain threshold (between 4 and 7, depending on the packet loss rate)), as less ACK packets

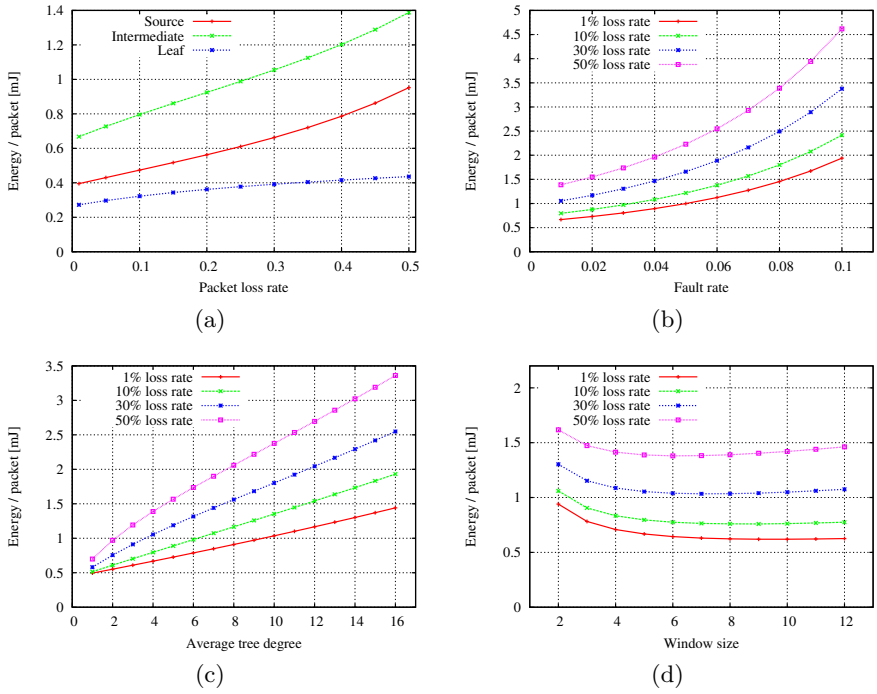
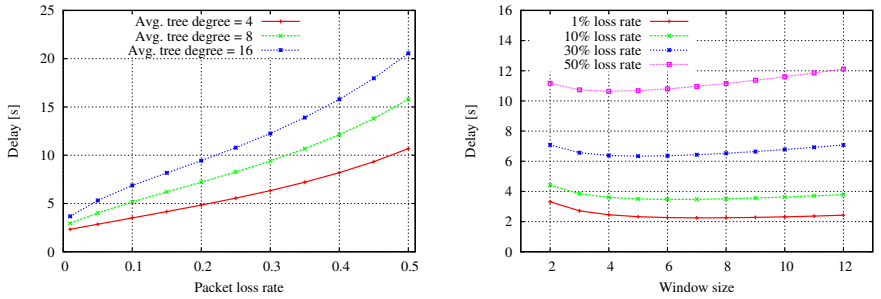


Figure 4.5: Theoretical results for energy consumption. (a) Source, intermediate and leaf nodes, as a function of the packet loss rate p_1; (b) Intermediate nodes as a function of the fault rate p_2; (c) Intermediate nodes as a function of the tree degree B; (d) Intermediate nodes as a function of the window size w.

are being produced. Beyond this threshold, the adverse effect of faults becomes prominent: within a larger window, faults determine the retransmission of more packets.

We further evaluate in Figure 4.6 the end-to-end latency of RMD, computed as the average delay per successfully transmitted and acknowledged packet. Figure 4.6(a) shows the average latency as a function of the packet loss rate p_1, for several values of the tree degree B. Figure 4.6(b) depicts the effect of window size w on the end-to-end latency. Similarly to energy consumption, the tradeoff value for w is between 4 and 7, and decreases with increasing packet loss rate (smaller windows are better for high loss rates).



(a) (b)

Figure 4.6: Theoretical results for end-to-end latency (a) as a function of the packet loss rate p_1; (b) as a function of the window size w.

4.6.2 Comparison

We compare via simulation RMD with PSFQ [167], a well-know reliable transport solution for WSNs. PSFQ is one of the first protocols that aims at providing reliable data dissemination with minimal energy expenditure in multihop WSNs. It has some similar properties to Scalable Reliable Multicast (SRM) [78], such as local recovery through request/repair packets and NACK suppression scheme. PSFQ is composed of three basic operations:

- *Pump operation.* The source node slowly injects messages at a constant "pump" rate into the network. Further on, relay nodes along the path rebroadcast each packet at this pump rate. Nodes listen for transmissions of the same packet in their neighborhood before rebroadcasting, in order to reduce redundant broadcasts.
- *Fetch operation.* Between two consecutive pump operations, nodes are offered the chance to ask for retransmissions of their missing packets (detected through gaps in the sequence numbers). Missing packets are indicated by sending NACK messages with the loss window in the local neighborhood. Any node hearing the NACK message and having the requested packets schedules a repair transmission. As in the case of regular packets, the number of redundant repairs is reduced by first listening for similar transmissions. The ratio between the pump and the fetch timers gives a number of possible NACK-based repairs that a node can request at each time step.

- *Report operation.* The report operation is intended to feedback data delivery status information to the source node. In order to minimize the number of feedback messages, nodes en route toward the source piggyback their report message in an aggregated manner.

In the simulation of PSFQ performed by Wan et al. [167], the scenario concerns the reprogramming of 13 nodes, regularly placed along the hallway of a building, with 50 packets of program data. We compare both protocols in a more general setting, using OMNeT++ simulation environment [21] (we reimplement PSFQ in OMNeT++ according to the description provided in [167]). We simulate a network of 100 nodes, randomly placed in an area of 200m x 200m, into which a source node attempts to disseminate 10000 packets. The radio range is set to 25 meters (as in the original simulation of PSFQ) and the channel error (packet loss rate p_1) varies between 1% and 50%. For every value of p_1, we average over 10 random topologies (the same for both protocols).

In the following, we discuss a number of design issues that make a fully objective comparison difficult. In each case, we explain the parameters used in our simulation.

Firstly, RMD relies on a tree structure. As shown in the previous section, the properties of the tree (degree, height) affect the overall performance. Since PSFQ is solely a transport solution, decoupled from the routing substrate, we are interested to compare only the reliable delivery portions of the two protocols. Therefore, we simulate RMD working on top of a simple hopcount-based spanning tree. To create the tree, at the beginning of every simulation run, each node selects randomly as its parent one of the direct neighbors with a lower hopcount to the source.

Secondly, PSFQ utilizes broadcast and NACK suppression, in order to reduce redundant broadcasts and avoid NACK implosion. As a result, nodes listen always to the medium, spending thus considerable energy on idle listening. In contrast, RMD keeps the radio off as much as possible through cross-layer listening (see Section 4.5.4). This advantage of RMD is not reflected in the simulations, as we only compare the energy spent on packet transmission and reception.

Thirdly, PSFQ does not explicitly handle faults as defined in Section 4.4.2. In case of hardware errors, RMD offers superior robustness. Nevertheless, in case of frequent link breakdowns PSFQ is in advantage, since no specific recovery is needed as long as the network does not become partitioned. This advantage of PSFQ is not reflected in the simulations, as we compare the performance of the two protocols only in presence of packet losses.

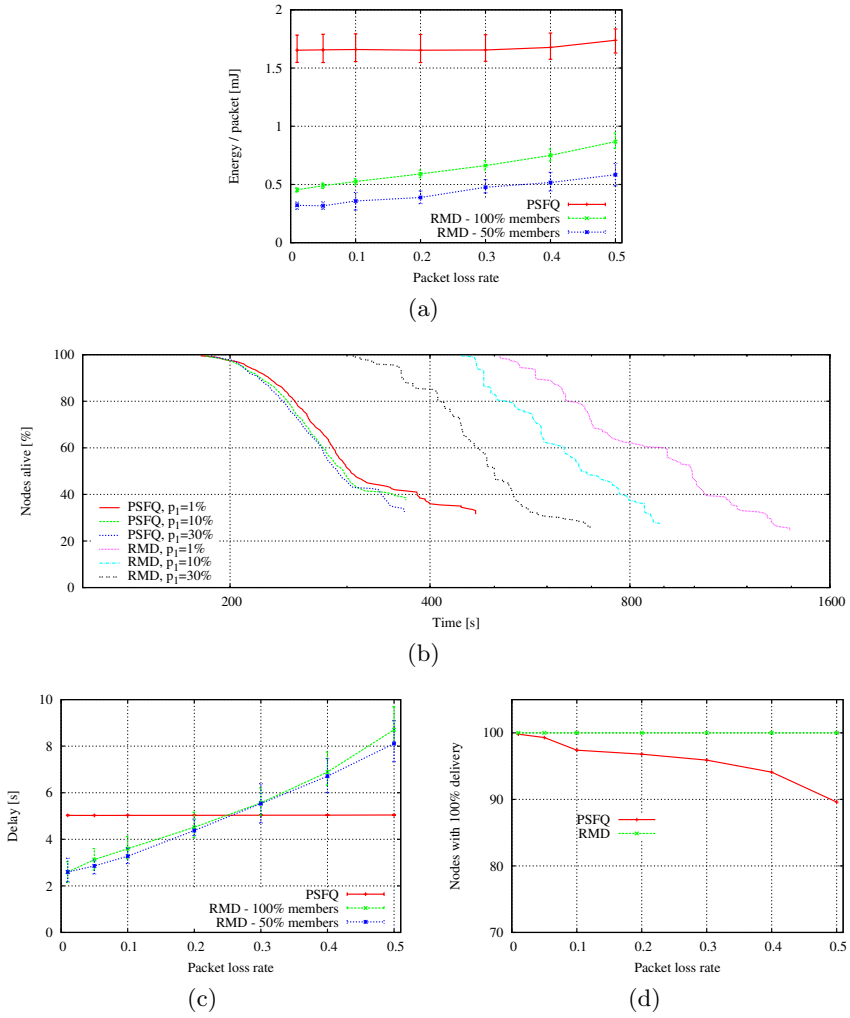


Figure 4.7: Simulation results for the comparison between RMD and PSFQ. (a) Average energy consumption as a function of the packet loss rate p_1; (b) Network lifetime (time axis at logarithmic scale); (c) Latency as a function of the packet loss rate p_1; (d) Delivery success.

Fourthly, the rebroadcast mechanism in PSFQ demands buffering along the way, otherwise the delivery ratio plummets with the error rate. There is no bound on the size of the buffers, which can exceed the available memory. We simulate PSFQ in an idealized case, with unlimited buffer sizes. Note that in RMD, the intermediate nodes maintain only a local cache of size w, comprising the current window of packets.

Finally, the timer values are important for the performance of both protocols. For PSFQ, we set the pump and fetch timers T_{pump}=1s and T_{fetch}=0.2s, which means 5 recoveries between two consecutive pump operations. For RMD, as mentioned, τ_1=1s. In this way, both protocols send packets at constant rate 1s.

Figure 4.7(a) compares the *energy spent on average by a node in the network*, under various packet loss rates. PSFQ incurs costs 2-3 times bigger than RMD, mainly because of the broadcast approach. An analysis over the data recorded from the simulation reveals that nodes running PSFQ consume 5-6 times more energy on receiving packets than on sending. Furthermore, if the multicast group does not include the whole network (which is an extreme case), the energy performance of RMD is even better, as some nodes remain passive (see Figure 4.7(a), RMD-50% members). Note that having 50% group members does not mean that the other 50% nodes are passive, as some of them may become forwarders at tree creation time.

The energy spent on average by a node provides only a partial performance metric, as RMD incurs different energy costs depending on the node role in the protocol (source, intermediate or leaf). Consequently, we also compare the *network lifetime* when running RMD and PSFQ. The plots from Figure 4.7(b) show the network lifetime at logarithmic scale, until the moment when less than 25% of the total nodes are still alive. Due to the reduced power network traffic, RMD clearly achieves a longer lifetime, outperforming PSFQ with a factor between 1.5 and 3, depending on the packet loss rate.

Figure 4.7(c) shows the *average latency* as a function of the packet loss rate. Since PSFQ has a fixed rate pump operation, the delay is affected only by the number of hops the packets have to travel, and not by the loss rate. Conversely, RMD is influenced both by the tree height (between 5 and 10, with an average of 7.7, for the various simulated topologies) and the loss rate. As a result, the latency tradeoff point between the two protocols is at a packet loss rate of 25%.

The constant delay exhibited by PSFQ comes at the cost of *having nodes that do not receive the entire message*. This is shown in Figure 4.7(d). To cope with this problem, PSFQ introduces a periodic report operation, where the nodes send aggregate feedbacks to the source. In our opinion, this is only a partial solution, since (1) the report messages do not benefit from the hop-by-hop error

recovery and, therefore, have higher chances of being lost along the path, and (2) there is no support for performing local repairs by examining the report messages.

4.7 Performance Evaluation of the Complete Network Stack

In the previous sections, we analyzed the reliable dissemination decoupled from the routing substrate and we assumed all the wireless links characterized by similar error probabilities on the average (p_1 and p_2). In the following, we present more thorough performance results, including the impact of the routing layer and the radio link model.

4.7.1 Simulation Framework

For simulating the complete network stack, we developed a framework comprising the following components:

1. The radio link quality model, derived from field trial results.
2. The data-link layer, represented by LMAC, a lightweight TDMA MAC protocol [89].
3. The topology control, implemented by a simple protocol for tree construction (FixTree), which represents the routing support of the dissemination layer.
4. The reliable dissemination layer, where the RMD protocol works in conjunction with FixTree and LMAC.

We developed our simulation framework in Simulink, which is a widely used tool for modeling, simulating and analyzing dynamic systems. Being integrated with Matlab, Simulink offers direct access to all the analysis tools available in the Matlab environment. The primary interface is a graphical block diagramming tool and a customizable set of block libraries. A typical Simulink model consists of blocks organized hierarchically and connected by signals. The models are hierarchical, which facilitates both top-down and bottom-up approaches.

Figure 4.8 shows the Simulink model of the network stack. The functionality of the blocks is as follows. The *Node source* block generates the network topology (random or a regular structure, see Section 4.7.5) and forwards it to the

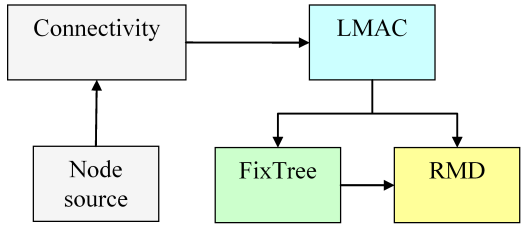


Figure 4.8: Simulink model of the simulation framework.

Connectivity block. The *Connectivity* block establishes the wireless links among the nodes and the link quality level, using data collected from a field trial (see Section 4.7.2). The *LMAC*, *FixTree* and *RMD* blocks implement the three communication protocols and interact with each other by exchanging matrices with relevant information.

In the *LMAC* block, the nodes first compete for occupying a timeslot, then they go into LMAC normal operation mode (sleep state for most of the time, except the periodic control messages) and wait for messages to be transmitted or received. After the initialization of LMAC is completed, the *FixTree* block activates the root node to build a tree, as described in Section 4.7.4. Eventually, the *RMD* block uses the tree to disseminate a message to all the nodes in the network.

4.7.2 Link Quality Model

In our simulations, we are interested to study the behavior of the communication protocol stack under a realistic model of the wireless link quality. More specifically, we analyze the change of the link quality with time and distance. For this purpose, we use the results of the field trial performed by Zhang et al. [187], where 18 sensor nodes were deployed at our university campus. The nodes were placed in different types of environments (in trees, within a parking lot and on building exterior walls). The trial lasted for 24 hours. In order to estimate the link quality, each node counted the LMAC control messages (see Section 4.7.3) received from every neighbor over a period of 256 frames. It follows that a value of 255 for the link quality indicates a very stable connection, while a 0 means no link at all. Even using this simple metric, the nodes had not enough storage capacity to keep the data locally, but transferred it periodically to a sink node connected to a PC. The resulting log file contains detailed information about

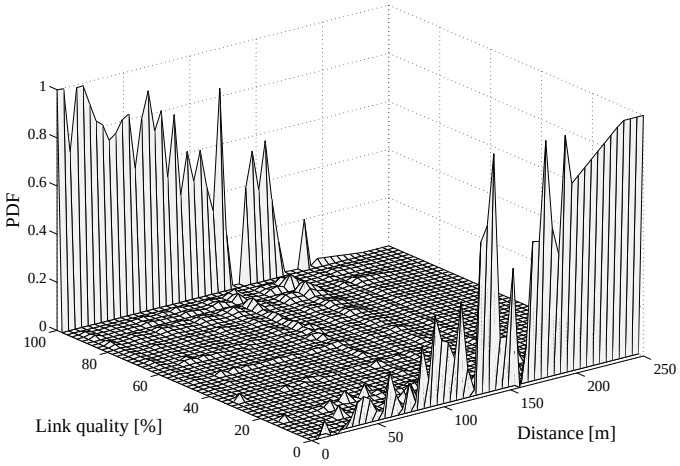


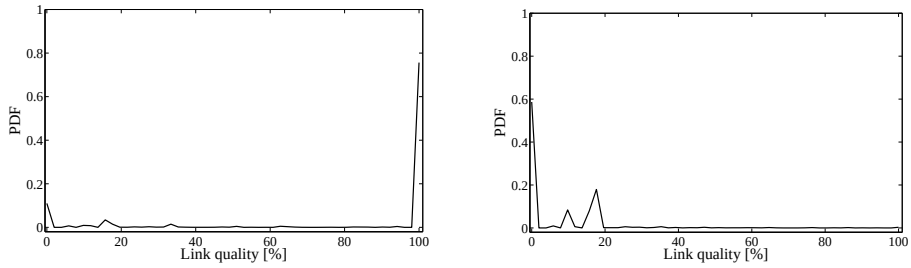
Figure 4.9: PDF of link quality obtained from the field trial in [187].

the evolution of the link quality in time between each pair of nodes, i.e. for 153 distances (note that since this was a multihop deployment, there were pairs of nodes that never had a direct connection).

In order to derive a link quality model for our simulator, we have to extract the following information from the results of the trial:

1. What is on average the initial link quality $q_0(d)$ between two nodes placed at any distance d from each other?
2. How does this link quality evolve in time, in other words, how do we predict $q_{t+1}(d)$ based on $q_t(d)$?

For the first problem, we use the procedure of building statistical models for lossy links in WSNs described by Cerpa et al. [58]. The probability that the link quality q_0 occurs at distance d is given by the amount of log entries found for the corresponding combination (d, q_0). The probability distribution function estimated through this method is represented in Figure 4.9, as a function of the distance among nodes and the recorded link quality (given in percents). We can notice the clear tendency of the links to be either very good (for distances smaller than 100m) or very bad (for distances larger than 175m). For clarity,



(a) (b)

Figure 4.10: Cross-section PDF of link quality from Figure 4.9: (a) for 50m distance, (b) for 180m distance.

we show in Figure 4.10 two detailed cross-sections of the PDF, corresponding to distances of 50m and 180m, respectively. Although this "binary" behavior seems extreme compared to results of related work [108], it shows that outdoor multihop WSN deployments *may* exhibit links with packet success rates either close to 0 or to 100% (with respect to [108], the "knee" value area may be very narrow). [5]

The second problem involves a more laborious procedure. For every distance d, we compute a stochastic matrix where the element at row i and column j gives the probability that the link quality $q_{t+1}(d)$ is j knowing that $q_t(d)$ was i. The following example helps clarifying this procedure. Suppose that for a certain distance d we have the following log fragment:

```
Time:     10  20  30  40  50  60  70  80  90 100 110 120 130
Quality: 255 255 255 255 245 255 255 255 255 255 145   0  10
```

We notice that link quality 255 is followed by 145 (1 occurrence), 245 (1 occurrence) and 255 (6 occurrences). Then, the stochastic matrix m^d corresponding to distance d has $m^d_{255,145} = 0.125$, $m^d_{255,245} = 0.125$ and $m^d_{255,255} = 0.75$. In the same way we can compute the values for link qualities 145, 245, etc.

[5] Especially in harsh industrial environments, the networking protocols should account for such phenomena when trying to ensure reliable data delivery. Moreover, the deployment should be carefully planned, as it can influence significant the overall performance (see Section 4.7.5).

4.7.3 LMAC Protocol

The LMAC protocol is based on scheduled access to the wireless medium. More specifically, time is organized in *frames*, and frames are subsequently divided into *time slots*. Each node within one-hop neighborhood gains control of one time slot every frame, in order to carry out its transmissions. For receiving messages, nodes always listen for a short period at the beginning of time slots, referred to as the *control message* period. The control message is followed by a much longer *data message* part, during which the node controlling the time slot can send its data. It is important to highlight that during the data message, only the intended receivers listen to the medium, while the rest of the nodes go into sleep mode. For the detailed presentation and analysis of LMAC, see [89].

To summarize, LMAC solves two important problems through scheduled access: avoids the problem of message collision and reduces the idle listening time. As a consequence, LMAC is shown to outperform the state of the art MAC protocols in terms of energy efficiency and delivery ratio [109].

4.7.4 FixTree Protocol

In our simulation framework, FixTree represents a baseline for a minimal routing protocol. It is designed to create a spanning tree over which the transport layer protocol disseminates the new data and code reliably to all receivers. FixTree does not support tree maintenance, so after the tree creation there are no further attempts to repair the broken links.

The description of the protocol is given in Algorithm 3. The standard hop-count tree formation approach is used: the source (or root) node floods an initial message in the network, and on the way back each node registers to a parent with a smaller hop count. To extend the lifetime of the tree, FixTree also takes into account link quality information, in addition to hop count, when choosing the parent node. As a result, the most stable links are preferred in building the tree.

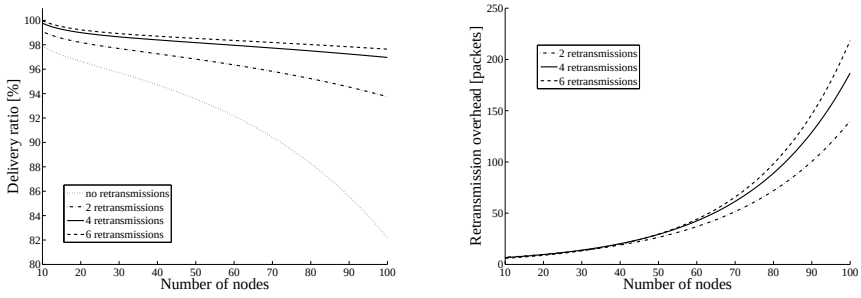
4.7.5 Results

In this section, we study through simulations the influence of different network properties on the overall performance of the reliable dissemination. The results plotted in the following figures are obtained by averaging over 40 simulation runs with different network topologies. Data is fitted using the Matlab curve fitting tool.

Algorithm 3: FixTree operation

Data:
u - current node ;
P - list of possible parents, initially $P = \phi$;
U - list of unregistered possible children, initially $U =$ one-hop neighbors ;
K - list of children registered to u, initially $K = \phi$;
$d(x)$ - hop count distance from x to source node ;
$q(x, y)$ - link quality between nodes x and y ;

```
// Packet reception
switch packet received from v do
    // Tree formation announcement
    case TREE_INIT
        if d(v) < d(u) then
            P = P ∪ {v} ;                              // add v as possible parent
            U = U − {v} ;        // remove v from unregistered possible children
        end
        if first time received and U ≠ φ then
            forward TREE_INIT;
        end
    // Registration request
    case PARENT_REGISTER
        if v wants to register to u then
            K = K ∪ {v} ;                          // add sender v to children list
            send ACK_REGISTER to v ;
        end
        U = U − {v} ;            // remove v from unregistered possible children
end
// Finish state decision
if U = φ then                                 // all possible children are registered
    if K = φ and u not in the multicast group then
        role is passive, do not join the tree ;
    else                                                  // register to best parent
        select parent p ∈ P with maximum link quality q(u, p) ;
        send PARENT_REGISTER to p ;
end
```



(a) (b)

Figure 4.11: Establishing the optimal number of local retransmissions in RMD, as a tradeoff between the delivery ratio (a) and communication overhead (b).

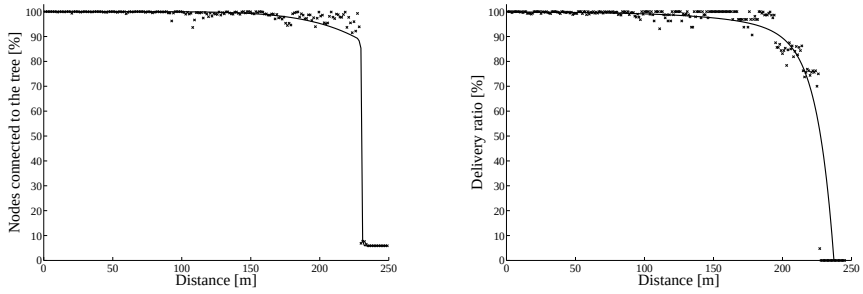
Experiment 1 - Local retransmissions in RMD

As a first step, we establish the optimal number of local retransmissions in RMD, as a tradeoff between the delivery ratio and communication overhead. We consider the nodes randomly deployed and vary the number of retransmissions from 0 to 6. Figure 4.11(a) shows that up to 4 retransmissions there is a significant increase in the delivery ratio (up to 15%). However, from 4 to 6 retransmissions, the improvement is less than 1% and, additionally, the communication overhead grows exponentially, as plotted in Figure 4.11(b). Consequently, we will use 4 retransmissions as an optimal value throughout the rest of our experiments.

Experiment 2 - Distance

In this experiment, we are interested to find out the impact of increasing distances between nodes over the performance of the network. For this purpose, we arrange the nodes in a hexagonal grid, so that the distance between each pair of nodes is the same. We are interested in two performance metrics: (1) the percentage of nodes connected in the tree by FixTree and (2) the percentage of nodes that received and acknowledged correctly the message through RMD (delivery ratio).

The results are plotted in Figure 4.12. We notice that FixTree manages to connect all the nodes in the tree for distances up to approximately 120m, and more than 90% of the nodes for distances up to approximately 175m. For distances larger than 230m the tree is practically impossible to build. Similarly,



(a) (b)

Figure 4.12: The performance of FixTree (a) and RMD (b) when varying the distance between nodes (hexagonal grid deployment).

RMD achieves more than 99% delivery for distances up to 120m, and more than 90% for distances up to 175m. For distances larger than 175m, the performance of RMD drops significantly. From this experiment, we can conclude therefore that 120m is a "safe" distance threshold for optimal performance of the network stack.

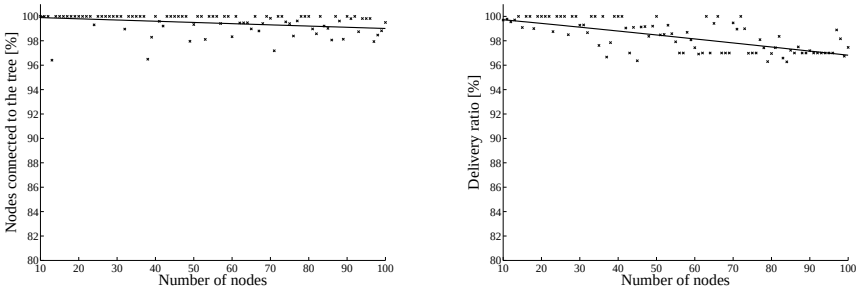
Experiment 3 - Network size

In this experiment, we evaluate the performance in networks with larger hop counts. The nodes are deployed in a star-grid topology. The distance between two neighbor nodes on the vertical and horizontal axis is 75m, and between two neighbor nodes on the diagonal axis is 106m, respectively. Since we are below the 120m threshold, distance is not a limiting factor in this experiment. By varying the number of nodes, we obtain average hop counts from 4 to 12 hops.

The results are plotted in Figure 4.13. We see that FixTree connects more than 99% of the nodes to the tree even for larger hop counts. Up to 50 nodes, RMD achieves a delivery ratio higher than 98.5%. For larger networks, the performance of RMD decreases linearly with the number of nodes and the hop count, getting to 96.8% for 100 nodes (12 hops).

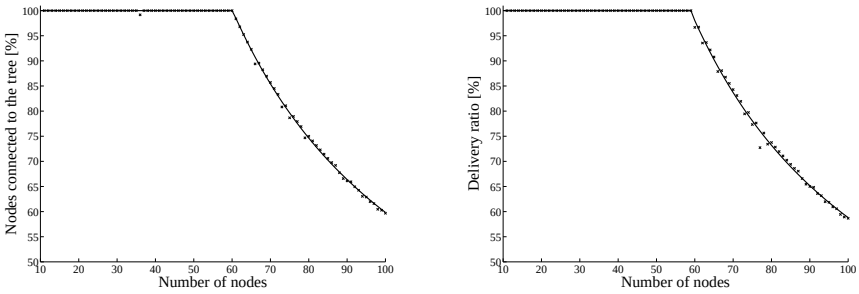
Experiment 4 - Network density

In this experiment, we study the impact of the network density over the protocol performance. We consider the nodes deployed randomly in an area of size



(a) (b)

Figure 4.13: The performance of FixTree (a) and RMD (b) when varying the network size and implicitly the hop count (star grid deployment).

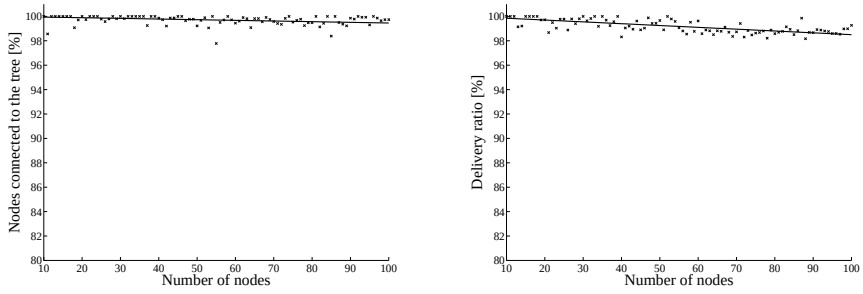


(a) (b)

Figure 4.14: The performance of FixTree (a) and RMD (b) when increasing the network density (random deployment).

250x250m. We progressively increase the density by increasing the number of nodes in the area from 10 to 100. Figure 4.14 shows the performance of FixTree and RMD, respectively. Since LMAC is a TDMA protocol, we observe a clear limitation after exceeding the number of available time slots. The nodes that cannot occupy the wireless medium are left out the dissemination tree and, even if they can receive the messages, they cannot acknowledge the correct reception.

From experiments 3 and 4 we conclude that the deployment of a WSN must be carefully considered, in order to make the best tradeoff between the number of hops and the network density. While the first factor affects linearly the delivery



(a) (b)

Figure 4.15: The performance of FixTree (a) and RMD (b) in the case of a planned deployment.

ratio, the latter poses a strict limitation to higher layer protocols.

Experiment 5 - Planning the deployment

As a final experiment, we analyze the situation of a carefully planned deployment, where the limiting factors identified so far are avoided. The results in Figure 4.15 show that for up to 50 nodes, almost 100% of the nodes are connected to the tree and receive and acknowledge the messages through RMD. For more than 50 nodes, the performance decreases very slowly. For 100 nodes, FixTree still connects 99.6% of the nodes to the tree and RMD delivers 98.8% of the messages correctly.

4.8 Prototyping

We implemented the RMD protocol on the Ambient μNode 2.0 platform [1], using the AmbientRT [91] real-time multitasking operating system and the LMAC protocol. According to our cross-layer design, the reliable data dissemination interacts with LMAC, in order to reduce idle listening and exchange local ACKs.

The first objective of our implementation is to prove the feasibility of running RMD on resource constrained sensor nodes. Table 4.4 lists the code and memory footprints of the implementation. As we can see, RMD consumes a minimal amount of both Flash memory and RAM, thus leaving enough space for the operating system, the rest of the network stack and the application.

	Code (FLASH)	**Data** (RAM)
RMD	2.7 kB	188 B
LMAC	3.6 kB	296 B
OS kernel and drivers	21.6 kB	400 B
Total available	48 kB	10 kB

Table 4.4: Code and data memory footprints

The second objective of implementing RMD is to validate experimentally the theoretical performance analysis from Section 4.5. For experiments, we use a small network of 11 nodes placed indoors (see also Section 5.3.2 of Chapter 5). In each experiment, RMD disseminates a message of 1kB from a PC to all the nodes. The communication between the PC and the gateway node is performed reliably using stop-and-wait ARQ. The multicast tree connections are preprogrammed into the nodes and varied accordingly as to experiment with different values of the tree degree B. At the end of each dissemination session, all the nodes report back to the PC the number of packets sent and received, as well as the number and type of errors recorded. Due to the limited scale of the experiment, the maximum observed packet loss rate p_1 is below 20% (compared to 50% as we used in the theoretical results from Section 4.6.1) and no faults occur (p_2=0).

Figure 4.16 compares the experimental and analytical results. The analytical results are obtained from the equations derived in Section 4.5. Figure 4.16(a) shows the average energy consumption on intermediate nodes under various packet loss rates, for a tree degree B=4. Figure 4.16(b) plots the dissemination latency for the same setting. Figure 4.16(c) presents a selection of results concerning energy consumption on intermediate nodes for various tree degrees, under the same packet loss rate p_1=5%. In general, the experimental results follow closely the analytical estimation. The energy efficiency is slightly better in experiments than in theory due to the impossibility of creating a sufficiently large pool of results for extracting the accurate probability p_1. Concerning latency, we measure additional delay (≈0.5s) in experiments compared to analytical estimation, for packet loss rates below 15%. The main reason for this result is that the analytical estimation approximates the protocol latency as being given solely by the delay on the longest branch of the tree.

The third objective of our implementation is to use RMD to re-task a multihop WSN at runtime. As described in Chapter 3, RMD was successfully demonstrated in the CoBIs project [4], by reconfiguring over-the-air groups of

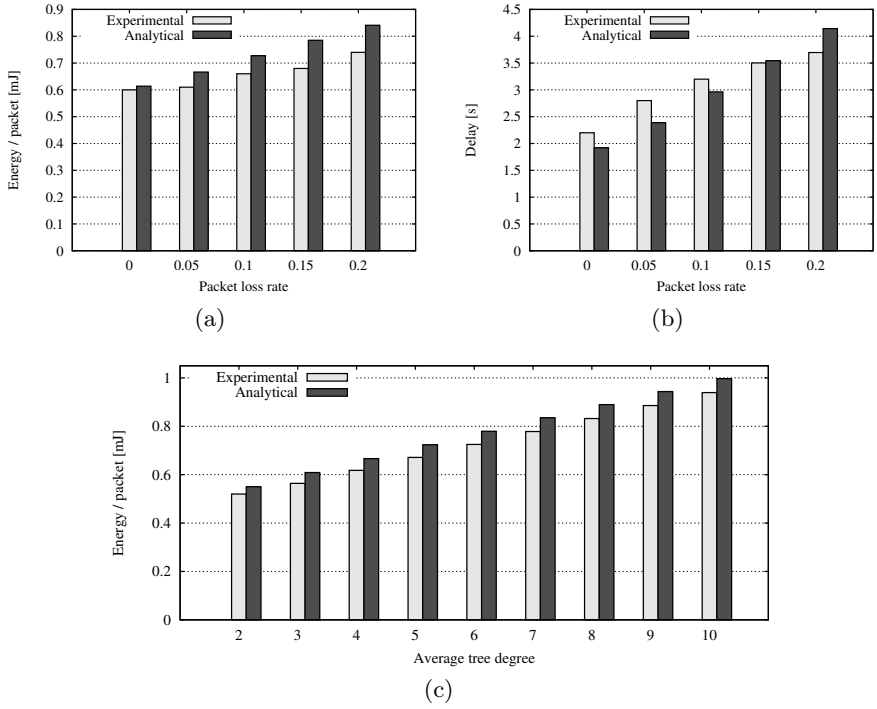


Figure 4.16: Experimental results compared to theoretical analysis. (a) Energy consumption as a function of the packet loss rate p_1; (c) Latency as a function of the packet loss rate p_1; (c) Energy consumption as a function of the tree degree B.

sensor nodes executing business rules. Subsequently, these experiments help us to observe the behavior of the integrated RMD – business rules solution in real conditions. For additional details, the reader is referred to Section 5.3.2 of Chapter 5.

4.9 Conclusions

Reliable transport remains an open issue in WSNs. A generic protocol, such as TCP stands for traditional networks, is expected to incur significant overhead compared to specialized solutions that optimize for a given traffic pattern (e.g.

event reporting, code dissemination, etc.) [177].

This chapter studied the problem of reliable multicast data and code dissemination from a central point to groups of sensor nodes. Based on initial prototyping with three basic methods of reliable transport, we designed RMD, a tree-based reliable multicast dissemination protocol. We provided an extensive study of RMD, including the theoretical performance analysis and simulation-based comparison to PSFQ, a representative reliable transport protocol for WSNs. Furthermore, we extended the performance evaluation to the complete network stack, composed of RMD, FixTree and LMAC protocols. At the physical layer we used a radio link quality model derived from field measurements. Finally, we presented the implementation details of RMD and experiences with comparing the theoretical analysis to the real protocol behavior.

With respect to the major requirements formulated in Section 4.1, RMD achieves the following:

- Ensures the data delivery to all recipients even under high error rates (packet losses up to 50%) and in harsh conditions (e.g. hardware resets).
- Due to the parent-children relation built among nodes, failures are always detected and reported upstream to the back-end system.
- Addresses groups of nodes instead of flooding the whole network. The user can reconfigure only a subset of the WSN, while the rest of the nodes remain passive and save energy.
- Achieves energy efficiency, by performing local error detection and repairs, and reducing the idle listening time. Compared to PSFQ, RMD consumes 2-3 times less energy while achieving a better delivery ratio.
- Keeps a low latency level, by pipelining the data along the tree. Compared to PSFQ, RMD latency increases linearly with the packet loss rate, with a tradeoff point at 25% packet loss.

From our experiences with implementing and testing reliable data dissemination on sensor nodes, the most important lessons learned are the following:

- Communication errors usually occur in burst and result in the loss of one or more packets. The transport layer should focus on detecting and repairing packet losses with minimal energy expenditure. Using MAC ACKs instead of explicit ACKs is an efficient solution to this problem.
- A minimal end-to-end control is essential for the back-end to have some guarantees on how the dissemination progresses in the network.

- Often in practice, the links among nodes are either very good or very bad. From the reliable dissemination perspective, only the good links are a viable choice for streaming a large message as fast as possible in the network.
- The large number of tunable protocol parameters (sending rate, window size, maximum number of retransmissions, tree degree) makes it difficult to find an optimal tradeoff among delivery ratio, energy efficiency and latency. Furthermore, the deployment properties (physical distance among nodes, density, number of hops) has a strong impact on the overall performance. The "best" values for all these parameters can only be established experimentally.

For future work, we suggest two main directions. First, *cluster-tree* networks represent a logical extension of the reliable multicast dissemination. The advantage of cluster-tree networks is that the more powerful nodes, connected through better links, can establish a stable spanning tree on which RMD maximizes its performance. In a subsequent step, each powerful node can act as a source for delivering the data within its one-hop neighborhood, using for example a receiver-initiated NACK-based protocol. The second direction of future work concerns the combination of reliable data dissemination and *multi-channel* communication. By making use of the already existing multiple-channel radios, sensor nodes can reduce both the delivery latency and energy consumption of reliable dissemination.

Chapter 5

Rule-Based Inference in WSNs

In this chapter, we study two rule-based inference engines through which WSNs can perform distributed situation assessment and event detection at the point of action. Based on the widely accepted business rule paradigm, we propose a lightweight business rule inference engine for WSNs, which offers the user the possibility to express easily the application logic and, moreover, to change it with minimal overhead. In a second step, we extend the flexibility and robustness of rule-based inference by utilizing fuzzy logic. The outcome is a distributed fuzzy logic inference engine that fuses sensor data and neighborhood observations to produce reliable results for the generic problem of event detection. This chapter is based on the following two publications: "Implementing Business Rules on Sensor Nodes" in International Conference on Emerging Technologies and Factory Automation (ETFA), September 2006 [125] and "D-FLER: A Distributed Fuzzy Logic Engine for Rule-based Wireless Sensor Networks" in International Symposium on Ubiquitous Computing Systems (UCS), November 2007 [124], which are joint work with P. Havinga. This work has been partially sponsored by the European Commission as part of the AWARE project (IST-2006-33579).

5.1 Introduction

The rapidly increasing interest of the industrial and business community in WSNs can be explained by two important factors. First, the comprehensive set of features available on current sensor nodes (digital I/O, storage and processing, wireless communication) makes real the term "intelligent sensor". Second, WSNs are self-organizing, collaborative networks, capable of executing logic in a distributed way, at the point of action. This is a desirable property because it decreases the load on the back-end system, improves the system reliability and responsiveness, and reduces the energy consumption.

Equally, there are two major obstacles on the way of making WSNs a state-of-the-art solution in industrial applications. The first one concerns the *programmability*. Despite the efforts of the research community, sensor node programming remains a tedious, low-level task. As a consequence, it creates unnecessary overhead, most notably in the case of what we call "rule-based applications". The use cases described in Chapter 3 represent such rule-based applications. They require (1) continuous, repetitive sensing, (2) detection of situations that do not comply to a set of user regulations and (3) alarming and local action. Because of cost concerns and their superior scalability and responsiveness, WSNs clearly constitute a suitable solution for these scenarios. What is lacking, from the user perspective, is the possibility of expressing easily the business logic behind the application and, moreover, of changing it with minimal effort.

The second difficulty stems from the inherent resource limitations of sensor nodes, dictated by the low cost and small form factor. In particular, sensor inaccuracy or faults can adversely affect the overall *robustness* of the system below the acceptable threshold for industrial applications. Sensor data fusion techniques alleviate this problem, but also require complex computations that sensor nodes cannot handle, or prior information that is infeasible to obtain practically [86].

In this chapter we address the two issues of programmability and robustness by exploring two rule-based inference methods for WSNs. Consistent to our methodology, we start with a prototype: the business rule engine for WSNs. This proved to be a very intuitive and compact programming model and was successfully demonstrated throughout the CoBIs project. In the next step, we extend the flexibility and robustness of rule-based inference by utilizing fuzzy logic. We design and evaluate D-FLER, a novel reasoning engine for sensor nodes, which combines the benefits of fuzzy logic with the distributed, collaborative nature of WSNs.

The rest of this chapter is organized as follows. Section 5.2 overviews the re-

lated work. The business rule concept is explained and evaluated in Section 5.3. A short overview of fuzzy logic is given in Section 5.4. Section 5.5 introduces the detailed design of D-FLER. Sections 5.6 and 5.7 present the performance evaluation considering a fire detection scenario. In Section 5.8 we analyze the implementation of D-FLER on real sensor nodes. Section 5.9 provides a brief discussion of the main advantages and limitations, and finally Section 5.10 summarizes the conclusions.

5.2 Related Work

Traditionally, WSNs were perceived as an enabling technology for data acquisition on a large scale. As a consequence, the efforts focused on simplifying the information extraction process. TinyDB [119] follows the idea of a declarative database for WSNs and provides a SQL-like query interface. The SQL syntax is extended with additional parameters, such as the data refresh rate. Given a query, TinyDB takes care of data collection, filtering, aggregation and routing from sources to sink. TinyDB runs on Mica motes and is included in the TinyOS distribution.

An application specific virtual machine (VM) offers more flexibility for programming WSNs, but at the price of significant overhead. Maté [113] is perhaps the most prominent work in the area. The Maté framework compiles programs to bytecodes and the VM runtime automatically propagates and installs the new code in the WSN. Maté is developed under TinyOS and supports TinyScript, a simple BASIC-like language, and mottle, a Scheme-like language [114]. Similar functionality is offered by SensorWare [51], a programming framework based on Tcl scripting language. SensorWare extends the Tcl command set with the necessary abstractions for script mobility, networking and sensing. In addition, SensorWare handles platform variability by borrowing the idea of virtual devices from Linux. The prototype platform is the iPAQ 3950, as the whole framework occupies 237kB of memory, out of which 64kB for tinyTcl and 35kB for the SensorWare core.

Previous research has also considered the idea of reasoning engines for embedded devices. Cooperative Artefacts [159] can autonomously reason about their situation by means of a Prolog-based inference engine. The artefacts maintain a shared knowledge base. The engine operates on rules and facts

represented as Horn clauses and uses a simplified backward-chaining [1] algorithm to prove a goal, essentially whether a query can be inferred from the facts and rules in the knowledge base. The prototype artefact is implemented on the Particle Smart-its platform and requires 1.2kB of code memory and 1.2kB of RAM. The functionality of the Prolog interpreter is, however, limited due to the limited resource available on the sensor nodes.

The ubiquitous chip [160] is an event-driven, I/O control device based on ECA (Event, Condition, Action) rules. The events are mapped to inputs from sensors or serial port, while the actions control the outputs of the device. The prototype ubiquitous chip contains a PIC16F873 microprocessor, 6 input ports and 12 output ports.

The two inference engines that we propose have common properties with the Cooperative Artefacts and the ubiquitous chip. The business rules exploit the simple IF-THEN structure (comparable to the ECA rule structure) in order to express service-oriented business logic in a WSN. The prototype is one of the most compact available, being implemented in less than 1kB of code memory on an MSP430 microcontroller. A step further, the D-FLER inference engine makes use of fuzzy logic to handle unreliable and imprecise sensor data, a feature not covered by previous work. As fuzzy inference preserves the simplicity of rule-based logic, the D-FLER implementation requires only 1kB of code memory.

5.3 Business Rules for WSNs

Business rules are nowadays a recognized paradigm for describing formally the operational constraints within an organization. The business rule sets can become surprisingly complex when applied to large organizational units. Example application domains include marketing strategies, pricing policies, customer relationship management practices, human resources activities and operation workflows [173].

Given the increasing uptake of WSN-based solutions into the industrial and business market, it is important to make the link between the sensor-level programming and the high-level business specifications. To fill this gap, we introduce the concept of *business rules for WSNs* as a simplified sensor programming interface.

[1] Backward chaining is a method of reasoning with inference rules, which works backward from the consequent to the antecedent. This method is goal-driven, in contrast to forward chaining, which is data driven. Backward chaining is often employed by expert systems and is supported in Prolog.

ID	Sensing driver	Sampling rate	Conditions (min, max, Δ)	Service	Running time	Next rule

Table 5.1: Structure of business rules for WSNs

5.3.1 Business Rule Engine Design

Business rules for sensor nodes represent a simple, yet powerful method of programming WSNs. The structure of the rules aims to express simple business logic in a compact and efficient way. Since sensor nodes assist real-life processes, the most common tasks focus on monitoring specific parameters against a set of conditions. If the conditions are not met or erroneous situations are detected, the nodes must inform the back-end system and take corresponding actions. A simple example is the following rule:

> Monitor temperature t and humidity h at rate r.
> If $t \notin [t_{min}; t_{max}]$ or $h \notin [h_{min}; h_{max}]$ or $\Delta t > \Delta t_{max}$,
> then start the climate control system.

Complex conditions can also be expressed by forming chains of rules logically linked and by providing more elaborate actions to be taken when the rules are fulfilled or not.

The structure of a business rule is defined in Table 5.1. Each rule is evaluated at the specified *sampling rate* by testing the values provided by the *sensing driver* against certain *conditions*. The driver usually represents the code that samples the sensed data, but it can be any function that provides a numerical result. In this way, richer functionality, computation or reasoning (e.g. obtain a consensus with the neighboring nodes) can be embedded in the rules. The conditions specify interval limits for minimum and maximum values, as well as the admissible variation Δ of the last sample compared with the previous ones. The evaluation of the rule outputs a TRUE or FALSE result that activates a *service*, i.e. executes a user defined code module. Moreover, each rule may be valid only for a certain *running time*, given by the start and stop moments. Finally, in order to construct chains of rules, a *next rule* field is provided.

Figure 5.1 gives an architectural overview of the business rule engine. The sensing infrastructure comprises not only environmental sensed data (such as humidity in the previous example), but also other contextual information that can be captured by sensor nodes and is relevant to the application (such as location information, neighboring nodes and even users, identified by the devices they are carrying with them). The rule engine is a standalone task, which

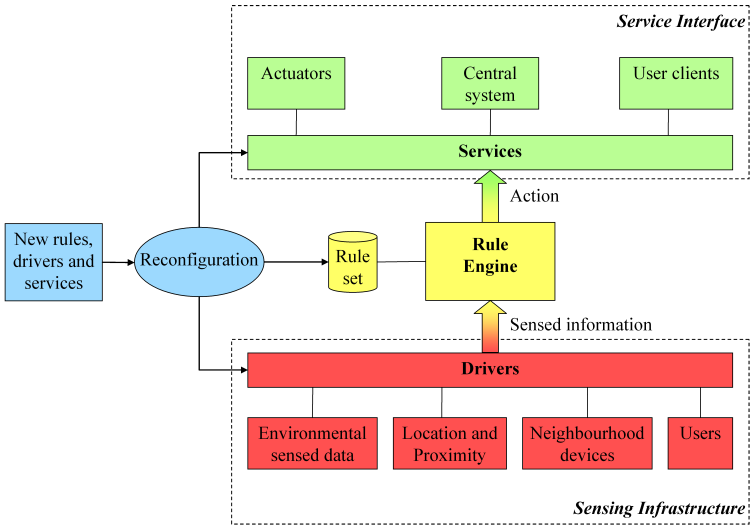


Figure 5.1: Business rule engine architecture.

evaluates the sensed information, provided by the drivers, according to the current set of rules. For every rule in turn, according to its sample rate, the rule engine calls the driver function, compares the data against the conditions and launches the appropriate service. The service can trigger a local action (such as a local alarm or an actuation), a remote connection to the central system, or even an interaction with a nearby user.

It is important to point out that the rule engine can handle both *polling* and *events*, depending on the sensing hardware available. For usual passive sensors, polling at the specified sample rate is the only option. In this case, the rule engine has the advantage that it triggers an action only when the sensed data does not match the conditions, saving thus network communication and energy, consequently. If active sensors are utilized (such as movement or vibration detection, push-buttons, or intelligent sensors that can be configured for low and high thresholds), then additional energy per individual node can be saved. The sample rate field can be ignored and the rule engine task is triggered directly by the drivers of the corresponding sensors.

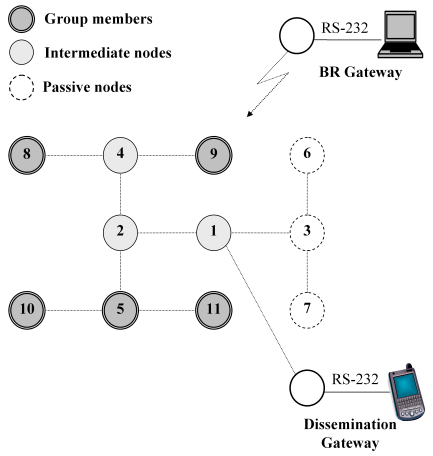


Figure 5.2: Experimental setting.

5.3.2 Practical Evaluation

We use for evaluation the multihop network of 11 nodes depicted in Figure 5.2. We have five nodes (5, 8, 9, 10 and 11) equipped with displays and the following sensors: internal voltage and temperature, light dependent resistor (LDR), temperature and humidity (SHT), and a push-button (see Figure 5.3). These sensor nodes execute continuously the set of business rules described in Table 5.2. Namely, they provide the following functionality:

- Alarm for maintenance in case of low battery level or excessive CPU internal temperature.
- Print the current running time on the local display.
- Monitor the light level, temperature and humidity, and alarm or regulate the climate control if these parameters get out of the acceptable range.
- Send a message to the server (BR gateway) when the user presses the push button.

The business rule engine is extremely compact, being implemented in less than 1kB of code memory. The rule set can be changed over-the-air from a mobile gateway composed of a sensor node connected to an iPAQ through serial interface. For disseminating the new rules we use the RMD protocol presented

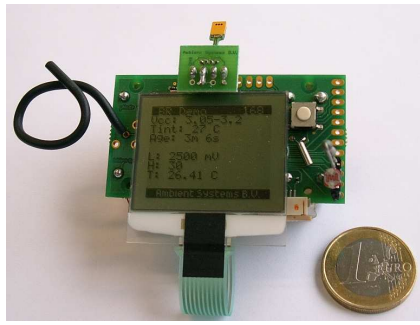


Figure 5.3: Sensor node executing the business rule engine.

in Chapter 4. From the perspective of RMD, the five nodes executing business rules form the multicast group, nodes 1, 2 and 4 are forwarding nodes needed for building the tree, and nodes 3, 6 and 7 are passive. When a new set of rules is announced in the initial phase of the dissemination, the sensor nodes enter a reconfiguration state. RMD takes the exclusive control of the radio to transfer the data as fast as possible. The rule engine task is "frozen" by periodically postponing its activation, similarly to a watchdog behavior. If any exception occurs and the dissemination does not terminate correctly, the business rule engine eventually resumes its execution. If the new business rules are safely transferred, RMD forwards them to the rule engine for reconfiguration.

We perform a first round of 30 experiments, in each disseminating a new set of business rules. In all the cases the nodes reconfigure correctly. The total disseminated data over all experiments cumulates $\approx$ 4kB. Each node stores logging information (such as the number of packets sent and received, the type and number of errors, etc.) and reports it back to the dissemination gateway. The user can thus follow on the iPAQ the ongoing process and an estimation of the energy consumption in the network

We now discuss the most important experimental results. The observed overall packet loss rate is 2.4%. Moreover, all lost packets are received properly at the first retransmission. This result points out that analysing MAC ACKs and performing local retransmissions represents a good choice. The average time for transferring one business rule is 2.7 seconds. This value is essentially given by the MAC frame length (1 second in our case) and the number of hops the data has to traverse. Figure 5.4(a) gives an overview of the average energy spent by each node for disseminating one rule. Node 0 represents the source,

ID	Name	Driver	Explanation	Service
1	V_{cc}	Internal	Battery level	Alarm
2	T_{int}	Internal	CPU temperature	Alarm
3	*Age*	Counter	Running time	Display
4	*Light*	LDR	Light level	Alarm
5	T	SHT	Temperature	Climate control
6	H	SHT	Humidity	Climate control
7	*Message*	Push-button	User message	Message

Table 5.2: Example set of business rules.

i.e. the dissemination gateway. As expected, the intermediate nodes (nodes 1, 2, 4 and 5) are the most loaded, being involved in forwarding both data and acknowledgments. The amount of energy spent on transmissions and receptions is balanced. In contrast, the leaf nodes and the source consume energy mostly on one operation (sending or receiving), according to their role. Finally, the nodes that are not group members (nodes 3, 6 and 7) remain passive during dissemination and save energy.

We perform a second round of 30 experiments, in order to test the behavior of the system under higher error rates. We produce the errors by randomly resetting nodes involved in dissemination and forcing in this way the protocol to recover from faults. The ratio of faults is on average 1.5%. This has also a marginal effect on the packet loss rate, which increases to 3.3%. It takes, however, at most two retransmissions to repair a packet loss. The most affected parameter is the average time per delivered business rule, which raises at 6 seconds. Figure 5.4(a) shows still reasonable values for the energy consumption; the increase computed over the whole network is $\approx 5\%$. This proves that the system has good ability of recovering from serious errors, even under high error rates.

5.3.3 Conclusions

Business rules for WSNs proved to be a successful programming model for industrial applications requiring distributed monitoring and event detection. Together with RMD, they were integrated in the service-oriented architecture demonstrated in the CoBIs project (see Chapter 3). Although conceived as an initial prototype, the business rules can fill the gap between low-level sensor programming and the business application. The business rule support is currently being implemented in real-life applications [1]. This result may be best

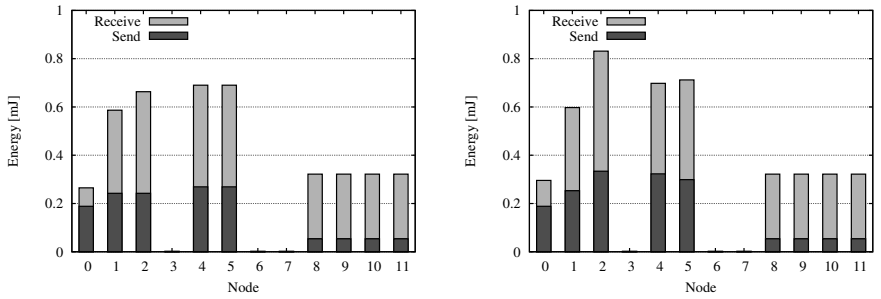


Figure 5.4: Experimental results of energy consumption during business rule dissemination (a) under normal conditions and (b) higher error rates produced by hardware resets.

explained by the well-known adage: "Sometimes, the simplest solutions are the best solutions".

The simplicity of business rules for WSNs implies inherently certain limitations. The most important concerns the lack of flexibility in describing complex situations. Consider for example a safety-critical event, such as fire, which can be inferred from various sources: temperature, smoke, gas concentration, etc. The recognition of such a complex phenomenon often involves sophisticated data fusion mechanisms, following the evolution in time of the combined parameters of interest. Using crisp logic and relatively simple conditional semantics based on interval limits, the business rule engine cannot ensure the desired level of inference. Erroneous and outlier sensor readings, which are likely to appear in case of a severe event such as fire, can easily produce wrong results when passed through the IF-THEN rules, by exceeding the predefined thresholds. In order to accommodate imprecise and vague sensor information, and to detect complex events more accurately and timely, we propose in the following a more advanced inference engine based on fuzzy logic: D-FLER.

5.4 Fuzzy Logic

The field of fuzzy logic has been developing for more than forty years, with many successful applications in diverse areas as automotive industry, artificial intelligence, medicine, behavioral science, just to mention some [75]. Fuzzy in-

ference systems (FIS) match two of the most challenging requirements of WSNs: (1) they are simple and can be executed on limited hardware, and (2) tolerate imprecise, unreliable data. In addition, FIS have several properties that are less mentioned in the WSNs literature, but are equally important from a practical point of view. First, fuzzy logic can reduce the development time compared with other techniques. In Bayesian calculus for example, prior probabilities need to be acquired by means of a statistical analysis requiring massive amounts of data [86]. With fuzzy logic, it is possible to have a running system by using only an intuitive, common-sense description of the problem. Second, fuzzy logic is flexible; it can be built on top of the expert knowledge, mixed with conventional control methods and easy to add or change functionality. Third, FIS are computationally fast [86], which is important because the processing capabilities of sensor nodes are limited. Fourth, FIS can be implemented with little memory overhead, which is a desirable property in WSNs because of (1) the limited memory on the sensor nodes and (2) the latency of the network reprogramming. Finally, FIS, potentially combined with neural networks, are adaptive, i.e. can be trained with examples or can learn at runtime from feedback.

In the following, we give give a very brief introduction of fuzzy logic and fuzzy inference. The interested reader is referred for details to the seminal works of Zadeh [184] and Mamdani [120].

5.4.1 Fuzzy Sets

Let us consider U the *universe of discourse*, which provides the set of allowable values for a variable x of interest. A usual *crisp set* C is defined as $C = \{x \mid x \text{ meets some condition}\}$. This definition can be expressed also by introducing the notion of *membership function* $\mu_C(x)$:

$$\mu_C(x) : U \rightarrow \{0, 1\}, \text{ where } \mu_C(x) = \begin{cases} 0 & \text{if } x \notin C \\ 1 & \text{if } x \in C \end{cases} \tag{5.1}$$

For a *fuzzy set* F, the membership function takes values in the interval $[0, 1]$. In other words, a fuzzy set generalize a crisp set in the sense that it allows its elements to have a certain *degree of membership* ranging from 0 to 1. By notation:

$$\begin{aligned} F &= \int_U \mu_F(x)/x, \text{ for } U \text{ continuous} \\ F &= \sum_U \mu_F(x)/x, \text{ for } U \text{ discrete.} \end{aligned} \tag{5.2}$$

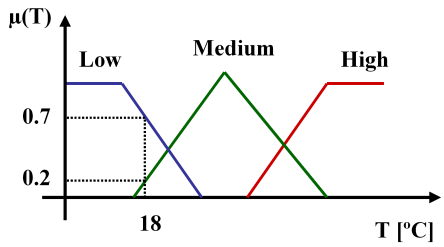


Figure 5.5: Membership functions for the linguistic variable *Temperature*.

Similarly to crisp set theory, fuzzy logic defines basic operations, such as union, intersection and complement. Let A and B be two fuzzy sets in U with membership functions μ_A and μ_B. Then:

$$\begin{aligned} \mu_{A\cup B}(x) &= max\{\mu_A(x), \mu_B(x)\} \\ \mu_{A\cap B}(x) &= min\{\mu_A(x), \mu_B(x)\} \\ \mu_{\bar{A}}(x) &= 1 - \mu_A(x). \end{aligned} \tag{5.3}$$

5.4.2 Fuzzification

Fuzzy sets form a systematic basis for working with *linguistic variables* [185], which are variables whose values are not numbers, but linguistic terms. Each term is characterized by a fuzzy set. Figure 5.5 gives an example of the linguistic variable *Temperature* (T), decomposed into three terms - *Low*, *Medium* and *High* - and the associated membership functions.

The first step in fuzzy inference is to map the crisp inputs (usually measured signals) to the corresponding linguistic variables through the use of membership functions. This step is referred to as *fuzzification*. Considering the example in Figure 5.5, a measured temperature value 18°C is fuzzified as 0.7 *Low* and 0.2 *Medium*. Often in fuzzy decision and control systems, both the current measured input and its differential variation Δ are fuzzified and taken into account in the inference. For example, assuming that the temperature is now 18°C and at the previous time step was 12°C, then the difference $\Delta = 6$°C may be fuzzified at 0.5 $FastRising$ and 0.8 $MediumRising$.

5.4.3 Rule Base

Fuzzy systems are fundamentally based on IF-THEN rules applied over the fuzzified inputs. Depending on the system to be designed, the *rule base* is constructed from expert knowledge, input-output data, or a combination of the two. The rules have the following format:

$$\begin{aligned} \text{IF} \quad & x_1 \text{ is } A_1 \text{ AND } x_2 \text{ is } A_2 \text{ AND } ... \ x_n \text{ is } A_n \\ \text{THEN} \quad & y \text{ is } B \end{aligned}$$

where x_i are input variables with their fuzzy sets A_i and y is the output variable with the corresponding fuzzy set B. Considering a fire detection system, an example of such a rule would be:

$$\begin{aligned} \text{IF} \quad & Smoke \text{ is } High \text{ AND } Temp \text{ is } Low \text{ AND } \Delta Temp \text{ is } FastRising \\ \text{THEN} \quad & FireDecision \text{ is } High \end{aligned}$$

5.4.4 Inference

In the *inference* step, each rule is interpreted as a fuzzy implication and the results of the rules are combined to produce an aggregate fuzzy output. Two of the most frequently used methods for implication and rule connection are *max-min* and *sum-product* [130].

Figure 5.6 illustrates the max-min fuzzy inference for the fire detection system already mentioned. In this example, the two input variables – temperature and smoke level – have measured values of 24°C and 52%, represented with vertical lines. Each input is fuzzified through three membership functions: *Low*, *Medium* and *High*. The rule base consists of four rules analyzing the fuzzified temperature and smoke values. The rules have to be "read" horizontally in the figure. In each rule, the minimum value between the two variables is selected and used to trigger the level of the output variable – fire decision – depicted in the rightmost column. The result of the inference is the aggregated output (i.e. the area built by taking the maximum point of the ouputs from all rules) shown in the bottom-right corner of the figure.

5.4.5 Defuzzification

The final step is to *defuzzify* the aggregated output back to a crisp number that can be used for making decisions or taking control actions. Possible defuzzification methods include: centroid method, largest of maximum (LOM), mean of

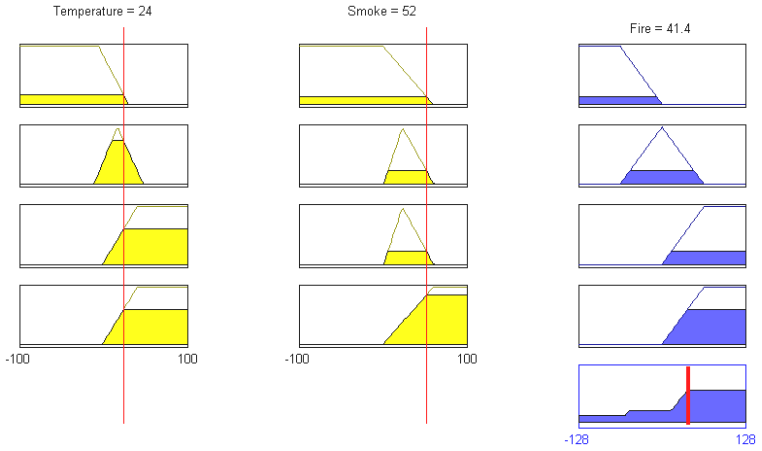


Figure 5.6: Fuzzy max-min inference for a simple fire detection system example.

maximum (MOM). In the example from Figure 5.6, the centroid method (i.e. the center of mass of the aggregated output area) yields the crisp result 41.4, which can be further classified as "potential high risk of fire".

5.5 D-FLER Design

WSNs usually try to compensate the resource limitations and the lack of reliability through cooperative algorithms, which exploit the high density of nodes deployed. This is why FIS for WSNs have to be distributed and to embed collaborative mechanisms of reasoning on the observed data and taking decisions or actions in a coordinated manner.

Following this idea, D-FLER uses two types of inputs: individual observations (sensor readings of the current node) and neighborhood observations (fuzzified sensor data from the neighboring nodes). Figure 5.7 illustrates how D-FLER fuses these inputs. We distinguish the following main operations:

1. *Fuzzification of individual observations.* D-FLER obtains the sensor readings from the sensor interface of the current node. Both the sensor raw values and their differential variations Δ are fuzzified through the predefined membership functions. The fuzzified values are scheduled for being broadcast within the local neighborhood by the MAC layer.

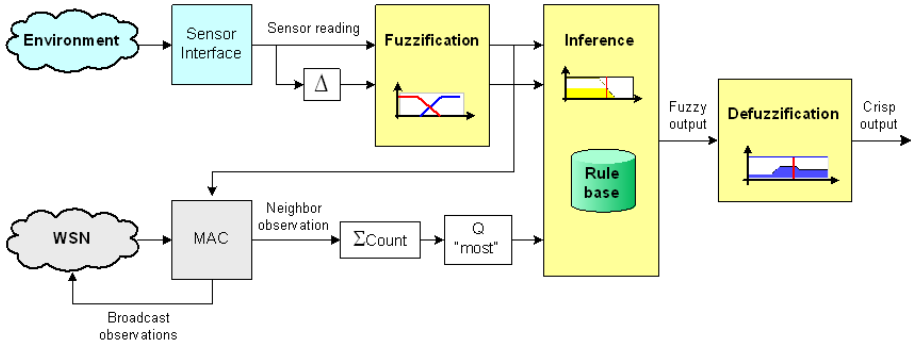


Figure 5.7: D-FLER structure.

2. *Quantification of neighborhood observations.* When receiving neighborhood observations over the radio, the MAC layer forwards them to D-FLER. Then, D-FLER updates the *sigma-count* factor [186], which is formally defined as:

$$\sum Count(F) = \sum_i \mu_F(x_i) \tag{5.4}$$

where $X = \{x_1, ..., x_n\}$ is the set of neighbors and F is a property of interest related to their observations, e.g. "smoke level is high".

Optionally, the neighborhood observations can be given weighting factors based on the confidence of each neighbor (e.g. given by the precision of its sensors, distance from the event observed, past accuracy computed as false alarm and rejection rates, etc.). In this case, we have a *weighted sigma-count*:

$$\sum Count(F; w) = \sum_i w_i \mu_F(x_i) \tag{5.5}$$

The neighborhood observations are eventually characterized through a fuzzy majority *quantifier*, such as *most* [99]:

$$\mu_{most}(\frac{\sum Count(F)}{|X|}) = \mu_{most}(\frac{\sum_i \mu_F(x_i)}{n}) \tag{5.6}$$

where

$$\mu_{most}(x) = \begin{cases} 0 & \text{if } x \leq 0.3; \\ 2x - 0.6 & \text{if } 0.3 < x < 0.8; \\ 1 & \text{if } x \geq 0.8. \end{cases} \tag{5.7}$$

The *most* quantifier gives a fuzzified indication of the consensual neighborhood opinion that the current node can add to its own observations, in order to take a more accurate decision. However, the decision of the node is not fed back into the neighborhood, in order to avoid an artificial increase of confidence due to ping-pong effects.

To better understand the fuzzy quantification step, consider the following simple example. Let us assume three nodes in the neighborhood, $X = \{x_1, x_2, x_3\}$, and F="smoke level is high". At the current time step, the three nodes report $\mu_F(x_1) = 0.5$, $\mu_F(x_2) = 0.8$, $\mu_F(x_3) = 0.8$. If the nodes have equal weighting factors, then the result of quantification is: $\mu_{most}(\frac{1}{3}(0.5 + 0.8 + 0.8)) = \mu_{most}(0.7) = 0.8$, which would be interpreted as "most of the neighboring nodes consider the smoke level high".

3. *Inference.* In addition to conventional fuzzy inference, the D-FLER rules incorporate both the fuzzified individual observations and the quantified neighborhood observations. Such a "distributed" rule has the following structure

$$\begin{aligned} \text{IF} \quad & s_1 \text{ is } F_{i_1} \text{ AND } s_2 \text{ is } F_{i_2} \text{ AND ... } s_p \text{ is } F_{i_p} \text{ AND} \\ & Q\, n_1 \text{ is } F_{j_1} \text{ AND } Q\, n_2 \text{ is } F_{j_2} \text{ AND... } Q\, n_q \text{ is } F_{j_q} \\ \text{THEN} \quad & o \text{ is } G \end{aligned} \tag{5.8}$$

where s_i are fuzzified sensor readings, n_j are neighborhood observations, Q is the majority quantifier, o is the output, F_{i_k,j_l} and G are input and output fuzzy sets, respectively.

Considering the fire detection example (see Section 5.6), a rule can be written as

$$\begin{aligned} \text{IF} \quad & Smoke \text{ is } High \text{ AND } Temp \text{ is } Low \text{ AND} \\ & most(SmokeNeigh) \text{ is } High \text{ AND } most(TempNeigh) \text{ is } High \\ \text{THEN} \quad & FireDecision \text{ is } High \end{aligned} \tag{5.9}$$

During the inference process, several rules are activated and contribute to the combined fuzzy output. Optionally, the combination of rules can be weighted according to the degree of belief to each rule, if such information is available.

4. *Defuzzification.* The last phase, defuzzification, produces a crisp output from the aggregate fuzzy output, as explained in Section 5.4.5.

In addition to these operations, D-FLER can be trained and can learn at

runtime if feedback is available. To achieve this, two parameters can be adjusted:

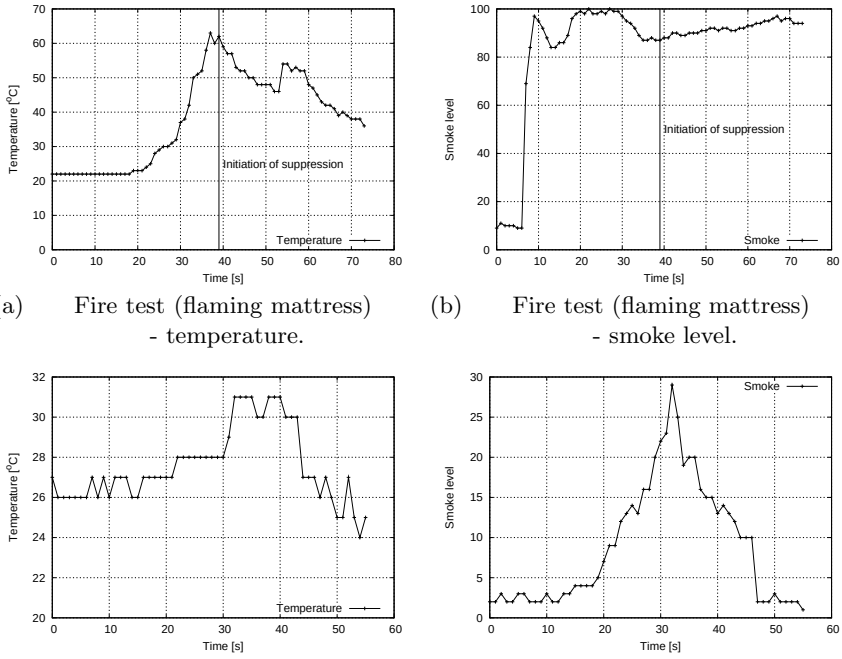
- The confidence of each neighbor in the *weighted sigma-count* factor. For example, nodes that constantly report observations in contrast with the right decision will receive a lower weight in the result of the quantification.
- The importance of each rule in the inference process. For example, after training, the rules contributing to correct decisions will be assigned a larger weight in forming the combined fuzzy output.

5.6 Fire Detection with D-FLER

We present a large-scale fire detection system as an application example for evaluating D-FLER. The potential usage of WSNs for real-time fire detection and firefighting assistance is currently under investigation in several research projects [2, 13]. Advanced fire detection algorithms consider distributed sensing as an alternative for improving time to alarm over single-station detectors [64]. In addition, comprehensive experiments show that the use of combined sensors [82] (e.g. smoke and CO sensors) can significantly reduce the false alarm rate, while increasing sensitivity (i.e., decreasing the detection time for real fires). Since fuzzy logic systems can fuse naturally multi-sensor data, they appear as a promising solution for robust fire detection algorithms [65].

In the following, we show that a distributed FIS, running within the WSNs, can improve the overall detection time and reliability, while providing better coverage for monitoring large hazardous areas. We utilize as input data the fire tests carried on by Bukowski et al. [54] to evaluate the performance of modern residential alarms. Both the fire and non-fire (i.e., false alarms, also referred to as nuisance alarms) test data is publicly available on the NIST website and well documented. We consider four representative tests: two fire scenarios (flaming mattress and flaming chair) and two nuisance scenarios (fried hamburgers and toasted bagel halves). For the fire scenarios, we follow the temperature and smoke data for approximately one minute after the moment of ignition. Similarly, for the nuisance scenarios, we follow the temperature and smoke data for approximately one minute around the time when the smoke alarms used in the tests reach medium alarm thresholds.

In Figure 5.8 we plot two example data sets for a fire and a nuisance scenario, respectively. Figure 5.8(a) and 5.8(b) show the temperature and smoke data for the first fire test used in our simulations (test SDC05 - flaming mattress in bedroom [54]). The ignition takes place at time 0. We notice that the



(a) Fire test (flaming mattress) - temperature.

(b) Fire test (flaming mattress) - smoke level.

(c) Nuisance test (toasted bagel) - temperature.

(d) Nuisance test (toasted bagel) - smoke level.

Figure 5.8: Example of fire and nuisance test data.

temperature increases much slower than the smoke level, which raises abruptly after 5s and reaches a maximum at 9s. However, the temperature has also to be monitored for a reliable detection because smoke can also occur in the case of a failed ignition. The temperature starts to raise quickly after 29s and reaches a maximum at 38s, right before the initiation of suppression.

Figure 5.8(c) and 5.8(d) show the temperature and smoke sample data for the second nuisance test used in our simulations (test MHN20 - two frozen bagel halves toasted [54]). During this test, the frozen bagels were toasted to a medium brown color and did not char significantly. We notice an approximately 5°C rise in temperature, with a peak between 32s and 40s. Similarly, the smoke level increases to a peak level until 32s. After stopping the toaster, the conditions come back to normal at approximately 48s.

The sensor nodes are assumed to sample the temperature and smoke values under two simple fire models:

- *Basic model.* All the nodes deployed on the area measure the same values at the same time, with the variations introduced by their different accuracy and measurement errors.
- *Radial model.* The fire spreads from a central point in circular rings with a specified speed. The nodes therefore measure the fire parameters shifted in time, according to their position on the area.

In case of fire, it is desirable that the large majority of nodes within the event radius report fire in the shortest possible time. Likewise, in the presence of nuisance conditions, a correct behavior minimizes the number of false alarms. Therefore, we take the percentage of nodes agreeing upon the real event status (fire/non-fire) as a measure of reliability, and the time within which the percentage converges to 100% as a measure of responsiveness.

We are now ready to introduce our simulation model and present the performance results.

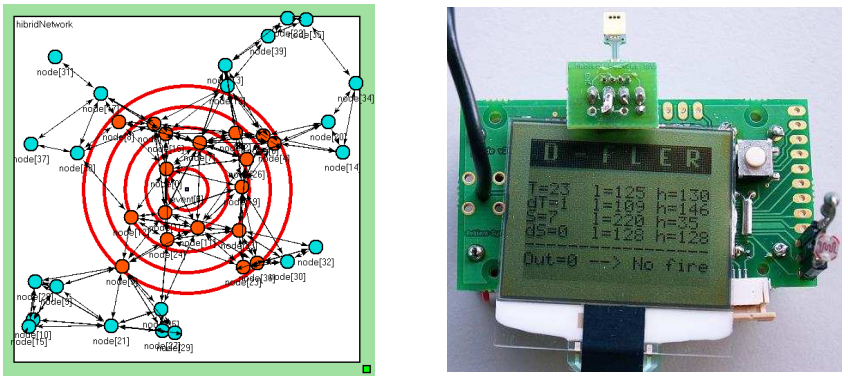
5.7 Simulation Results

5.7.1 Simulation setting

We consider a random deployment of 100 nodes within a rectangular area of 500m x 500m. The radio range is set to 150m. The nodes are considered to be randomly equipped with temperature and/or smoke sensors. The temperature sensor data is expressed in Celsius degrees. The smoke sensor data is derived from the ionization smoke alarm analog output used in the NIST tests, and scaled to 100. The accuracy of the sensors is modeled according to the characteristics of a real sensor [19], by taking into account the linearity, offset and gain errors, plus Gaussian white noise. The resulting sensor accuracy lies in the interval ±1–4%.

We implement the following four detection methods in the OMNet++ simulation environment [21] (see Figure 5.9(a) for a graphical image of the simulation setting):

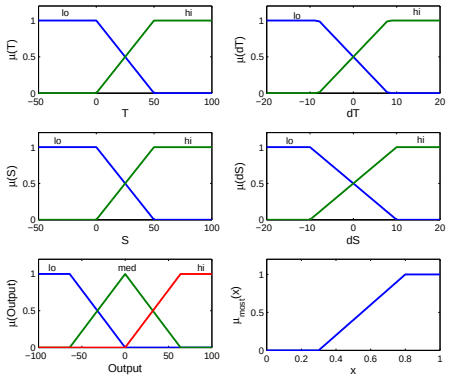
1. *Threshold.* Each node decides that a fire occurred when the temperature and smoke values exceed predefined thresholds. The thresholds correspond to the activation time of the medium alarm level in the NIST fire tests.



(a) OMNeT++ simulation. (b) D-FLER on the μNode platform.

Figure 5.9: D-FLER simulation and prototype.

2. *Average.* A simple distributed mechanism is implemented, where each node takes the mean value between its own samples and the average of the readings reported by its neighbors. The decision is made according to the same thresholds as in the previous case. This method is expected to decrease the false alarm rate in the case of nuisance scenarios, by reducing the effect of individual sensor errors.
3. *Fuzzy.* Each node uses a local fuzzy logic engine. If the node has both temperature and smoke sensors, then four inputs are analyzed: T - temperature, S - smoke level, dT - temperature change (current sample minus previous sample) and dS - smoke level change (current sample minus previous sample). For the nodes with only one sensor, the fuzzy logic engine takes two inputs (T and dT or S and dS). The membership functions are shown in Figure 5.10(a). Similar to the first method, no information is exchanged among the nodes.
4. *D-FLER.* As described in Section 5.5, the nodes broadcast their fuzzified temperature and smoke values within the local neighborhood. D-FLER operates with both the inputs mentioned in the previous method (T, S, dT and dS) and the quantified neighborhood observations. The *most* quantifier defined in Eq. 5.7 is used (see Figure 5.10(a)). The rules have the format described in Eq. 5.8 and 5.9 and are partially shown in Figure 5.10(b).



(a) Membership functions and μ_{most}.

Rule	T	dT	S	dS	most(T_n)	most(S_n)	Out
1	LO	LO	LO	LO	LO	LO	**LO**
2	LO	LO	LO	LO	LO	HI	**LO**
3	LO	LO	LO	LO	HI	LO	**LO**
4	LO	LO	LO	LO	HI	HI	**LO**
5	LO	LO	LO	HI	LO	LO	**LO**
6	LO	LO	LO	HI	LO	HI	**MED**
7	LO	LO	LO	HI	HI	LO	**MED**
8	LO	LO	LO	HI	HI	HI	**MED**
9	LO	LO	HI	LO	LO	LO	**LO**
10	LO	LO	HI	LO	LO	HI	**LO**
11	LO	LO	HI	LO	HI	LO	**MED**
12	LO	LO	HI	LO	HI	HI	**MED**
13	LO	LO	HI	HI	LO	LO	**MED**
14	LO	LO	HI	HI	LO	HI	**MED**
15	LO	LO	HI	HI	HI	LO	**HI**
16	LO	LO	HI	HI	HI	HI	**HI**
...	...	...	...	...	...	...	...

(b) Rule set (partial).

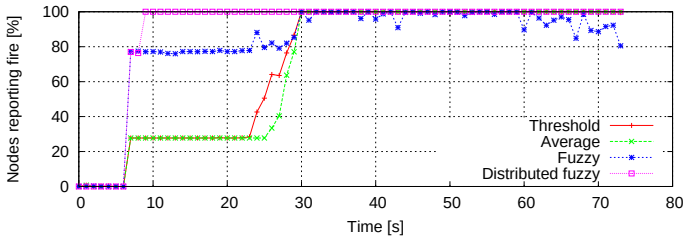
Figure 5.10: Fuzzy-logic fire detection engine.

5.7.2 Results

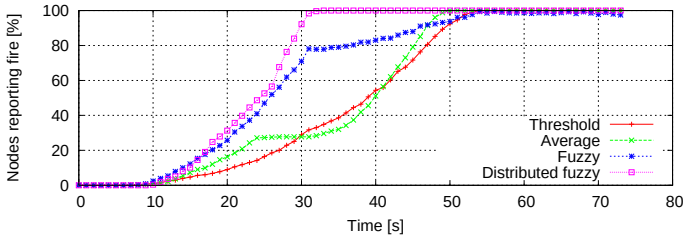
The four data sets from Figure 5.8 are input to the four simulated detection methods. We collect the decisions of the nodes at every time step. The results presented in Figure 5.11 and 5.12 are averaged over 10 simulation runs with different random topologies.

In the case of fire, we are interested in a rapid and reliable detection. The more nodes detect the fire, the more reliable is the decision on the average case and the information about the event can be transported faster toward the sink or gateway nodes. Figure 5.11 plots the percentage of nodes reporting fire during the two fire tests, considering both the basic and the radial spread model. We can make the following observations:

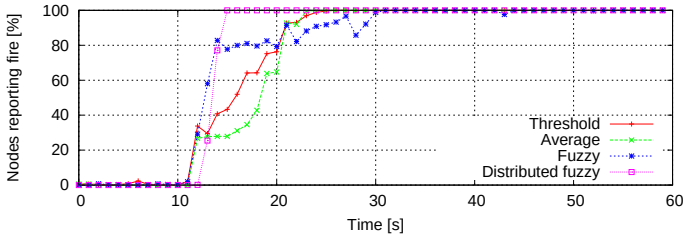
- D-FLER achieves the shortest detection times both for the basic and radial fire models, followed by the individual fuzzy engine. This indicates that fuzzy logic can model with more granularity the event of interest and provide a more robust inference result.
- The individual fuzzy engine has a few decision oscillations (especially noticeable in Figure 5.11(a)), while D-FLER has a clear transition from non-fire to fire state in all the situations. This shows that the individual reasoning method is more sensitive to sensor errors than the distributed one.



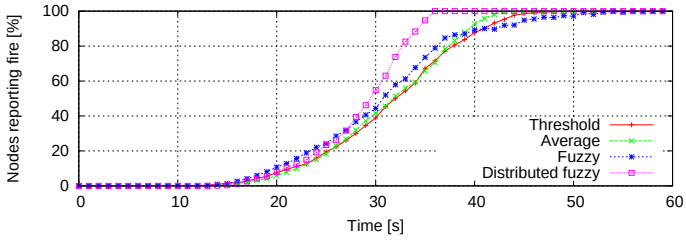
(a) First fire test - basic model.



(b) First fire test - radial model.

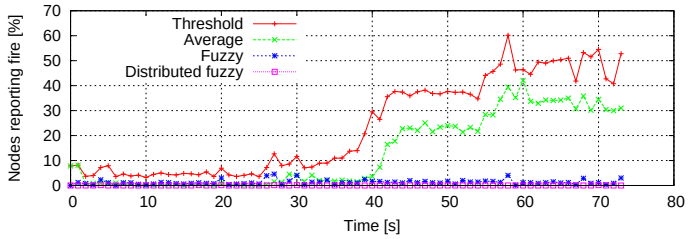


(c) Second fire test - basic model.

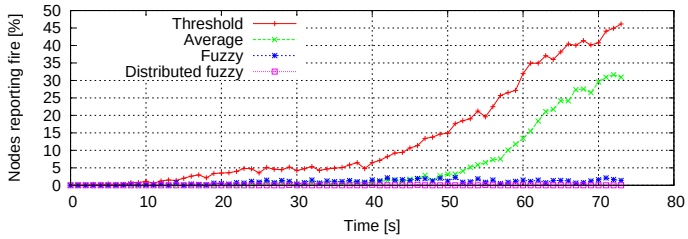


(d) Second fire test - radial model.

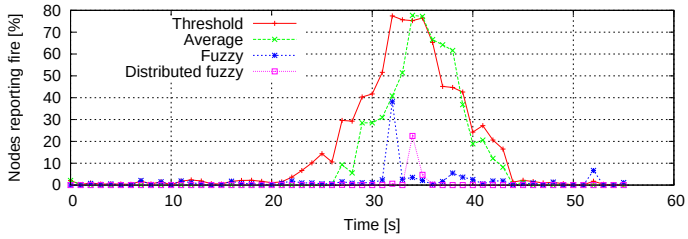
Figure 5.11: Simulation results of fire tests.



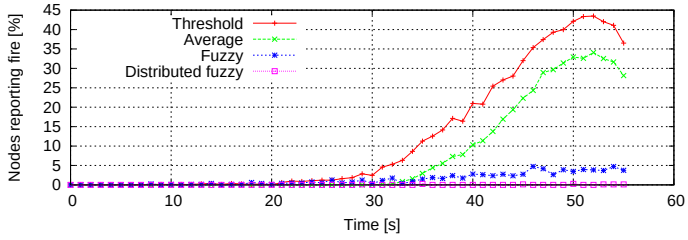
(a) First nuisance test - basic model.



(b) First nuisance test - radial model.



(c) Second nuisance test - basic model.



(d) Second nuisance test - radial model.

Figure 5.12: Simulation results of nuisance tests.

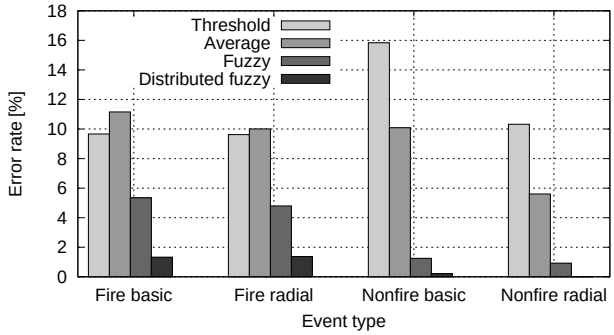


Figure 5.13: Summary of simulation results - error rates of the four methods.

- The average method is slightly slower to converge to 100% nodes reporting fire than the direct threshold method because of the slow temperature raise and the influence of the neighbors samples.

In the case of nuisance tests, we are interested in reducing the false alarm rate. Figure 5.12 presents the simulation results of the two nuisance scenarios. We can make the following observations:

- The average method reduces the false alarm rate compared to the direct threshold method, proving to be less sensitive to individual sensor errors.
- The fuzzy logic-based methods are clearly more robust to nuisance conditions, with D-FLER approaching to 0% erroneous nodes.

Figure 5.13 gives a summarized view of all the simulation results. The error rate is computed as the percentage of erroneous decisions, over all the nodes and the entire simulation duration. The fuzzy logic methods prove to be more reliable than the threshold solutions, D-FLER having an average error rate less than 2% in the case of fire and approaching 0% in the case of nuisance tests.

5.8 Prototyping

We implemented D-FLER on the Ambient μNode 2.0 platform [1] (see Figure 5.9(b)). The μNodes run AmbientRT [91], a real-time multitasking operating system based on publish/subscribe inter-task communication. The publish/subscribe model simplifies the implementation, as D-FLER can abstract

from various types of inputs by subscribing only to their data. Similar to the business rule engine, D-FLER is decoupled from the *drivers* (custom interfaces giving the sensor readings or radio protocol stack delivering the neighbors messages) that provide the input data. The system is therefore modular and easy to reconfigure over the air.

The following list summarizes the implementation details of the D-FLER components (see also Section 5.5):

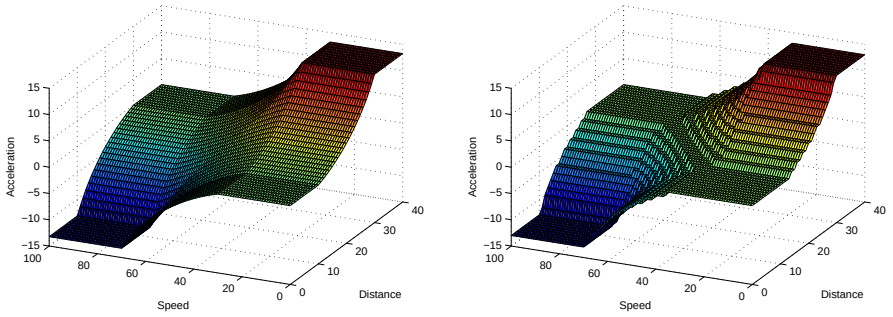
1. *Fuzzification.* The local sensor readings are fuzzified according to the user-specified membership functions. D-FLER currently accepts only triangular membership functions. Our implementation of the fuzzification is *computational oriented* [66] due to the small amount of RAM available on the node. To reduce the computational complexity, the maximum fuzzified value is scaled to a power of 2, as recommended by Dannenberg [67].
2. *Quantification of neighborhood observations.* The data from the neighboring nodes is processed through the *sigma-count* factor and μ_{most} operator (see Eq. 5.4 - 5.7).
3. *Inference.* The rules are formatted as in Eq. 5.8, taking into account all the fuzzified inputs. D-FLER uses *max-min* inference for computational simplicity.
4. *Defuzzification.* The aggregated result of the rule evaluation is defuzzified using the *centroid* method.

We evaluate the D-FLER implementation by following three properties of interest: the memory overhead, the numerical accuracy and the execution time.

The code memory footprint amounts to ≈1kB FLASH memory (out of 48kB available), leaving thus enough space for the OS kernel, sensor drivers, network stack etc. In addition, D-FLER occupies 20 bytes RAM for static variables and allocates dynamically heap space for the inputs, outputs and rules. To estimate the memory consumption M at runtime, we use the following formula

$$M = I(4m_i + 1) + NIm_i + 2R(Im_i + 1) + O(4m_o + 1) \tag{5.10}$$

where I is the number of inputs, each input having m_i membership functions, N is the number of neighbors providing same type of inputs (fuzzified), R is the number of rules and O is the number of outputs, each output having m_o membership functions. An important optimization can be made if every input has two membership functions ($m_i = 2$). In this case, the rules can be represented only as their consequence parts ("o is G" from Eq. 5.8). Then, the binary



(a) Matlab. (b) D-FLER.

Figure 5.14: The car control problem.

representation of the rule index gives the combination of inputs and fuzzy input sets in the antecedents. For example, a hypothetical rule number 5 would be represented as binary 101, meaning "i_0 is F_1^0 AND i_1 is F_0^1 AND i_2 is F_1^2". Using this optimization, a running instance of D-FLER with 5 inputs and 2 outputs allocates 326 bytes RAM out of 10kB available.

D-FLER performs the computations on two-byte signed integers. In order to evaluate the numerical precision, we run a simple car control problem with two inputs (distance and speed) and one output (acceleration). Figure 5.14(a) and 5.14(b) plot the rule surfaces generated with Matlab and D-FLER, respectively, where the latter is obtained by iterating through the whole input space with unit step. The absolute error of D-FLER (due to integer approximation) is 1% on average, with a maximum of 3.33%.

The execution time of D-FLER is depicted in Figure 5.15 (at logarithmic scale) for different problem complexities, given by the number of inputs, outputs and rules (the number of membership functions is fixed at $m_i = 2$ and $m_o = 3$). For each case, the execution time is computed by averaging over the entire input space, as in the previous example. We can make the following observations:

- The fuzzification and inference times are in the same range, varying approximately linearly with the number of inputs and rules, respectively. The fuzzification operation takes between 180 and 360μs, while the inference process takes between 80μs and 1.13ms.
- The defuzzification is clearly the most time-consuming operation (between 6.56 and 25.86ms) because of the computation involved in calculating the

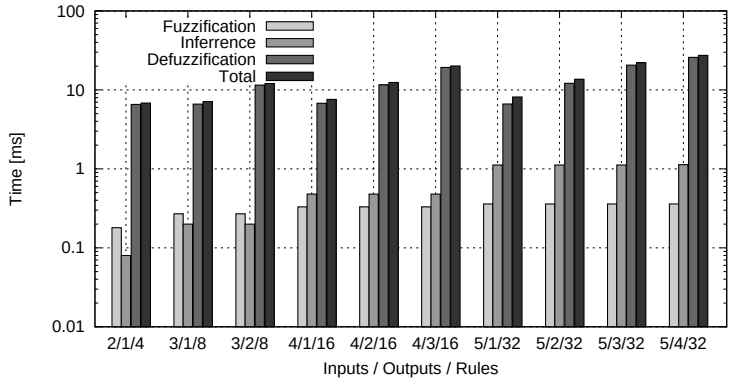


Figure 5.15: Performance of D-FLER on μNode platform (execution times at logarithmic scale). The MSP430 controller is set to operate at 4.6 MHz.

centroid of the output area. There are several alternatives to reduce the defuzzification time: (1) use rectangular output membership functions [67], (2) use a fast centroid approximation method [149] or (3) use a simpler defuzzification method, e.g. maximum.

- The total execution time is basically given by the defuzzification speed and, consequently, varies linearly with the number of outputs.

5.9 Discussion

In this section, we briefly discuss the impact of several factors on the performance of D-FLER, pointing out the main advantages and limitations.

Communication. Exchanging the observed data within the one-hop neighborhood increases the communication overhead and, consequently, the energy consumption. On average, each node would send one packet and receive $N\pi r^2/A$ packets per time step, where N is the total number of nodes, A is the area size and r is the radio range. Reducing the duty cycle alleviates the problem, but at the price of increasing the detection time. A better approach is to use cross-layer integration and piggyback the data to the periodic heartbeat messages of the MAC or routing protocols.

Sensing errors. In practical deployments, cheap sensors with low precision may be used due to cost concerns. Likewise, the phenomenon to be detected (e.g.

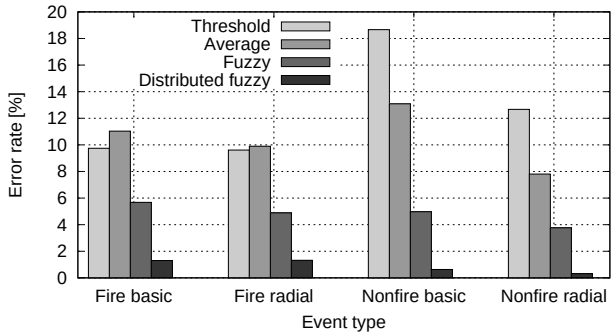


Figure 5.16: Error rates for sensors with lower accuracy.

fire) may itself introduce severe sensing errors. We repeated the simulations from Section 5.7 considering this time sensors with two times lower accuracy (2-8%). The summary results are shown in Figure 5.16. In the case of fire, there is little increase in the detection error rate (less than 0.3%) compared to the results from Figure 5.13. However, for the nuissance scenarios, the false alarm rate grows with up to 3.7% for all the methods except D-FLER, which stays lower the 0.5%. This result highlights the superior robustness to erroneous inputs of distributed fuzzy inference compared to the other methods. [2]

Computation. The fuzzification using triangular or trapezoidal membership functions and the max-min inference prove computationally fast in our implementation. However, the usage of other membership functions, such as Gaussian, would impose a memory-oriented implementation [66], less suitable for the low amount of RAM typically available on sensor nodes. The centroid-based defuzzification introduces the highest latency and should be optimized or replaced with a simpler method if execution times less than 6ms are needed. [3] The usage of faster or dedicated hardware is also an option, provided that the energy budget of the node allows it.

Training and learning. In order to produce optimal results, tedious manual tuning of the membership functions may be required. This can have two negative side-effects: (1) the optimal performance is not achieved and (2) the

[2]Further experiments confirmed that the difference in performance between D-FLER and the other methods increases when decreasing the sensor accuracy. However, this work does not provide a systematic quantitative analysis on this topic. For future work, see Section 5.10.

[3]Applying simpler defuzzification strategies may result in a loss of granularity or accuracy. We do not have, however, systematic results on this tradeoff.

performance is too sensitive to the configuration of tuning parameters. To alleviate these problems, standard methods such as ANFIS [98] (Adaptive-Network-Based Fuzzy Inference System) and FCM [47] (Fuzzy C-Means Clustering) can be used in order to systematically build a fuzzy model based on both human knowledge and stipulated input-output data pairs. In addition, since D-FLER is a distributed solution, learning at runtime about the confidence of the neighborhood observations would also be a valuable feature in real-life deployments.

5.10 Conclusions

The adoption of WSNs into industrial and business applications depends heavily on two factors: programmability and robustness. The user typically expects WSNs (1) to provide an intuitive programming interface that matches the logic behind the actual process and (2) to compensate for sensor inaccuracy and faults through the scale of their deployment.

This chapter explored the usage of WSNs in rule-based applications. There is a substantial set of such applications, ranging from classical event detection problems to business rules involved in transport and logistics processes. We designed and evaluated two rule-based inference methods for WSNs. As a first prototype, business rules proved to be a straightforward programming model and are currently being implemented in several large-scale applications [1]. The business rule engine we developed is one of the most compact solutions available in the WSN community, being implemented in less than 1kB of code memory.

The simplicity of business rules comes at the cost of certain limitations in terms of expressiveness and data fusion possibilities. D-FLER builds on these points and offers to the WSN user a distributed fuzzy logic reasoning engine. By combining individual sensor inputs with neighborhood observations, D-FLER produces more accurate results and is more robust to sensor errors. For performance evaluation, we used fire detection as an application scenario inspired by the EU projects CoBIs [4] and AWARE [2]. The results showed that distributed fuzzy logic is a promising alternative for event detection with WSNs, as it improves the detection time, while reducing the false alarm rate. From the implementation point of view, D-FLER proved effective and feasible to run on resource-constrained sensor nodes.

The main directions of interest for future work have already been sketched in Section 5.9. From the networking perspective, the tradeoff between the communication overhead and detection performance requires further investigation. On the sensing side, a generalized relation between the error level in sensor readings

and the detection performance would be an interesting result for characterizing formally the robustness of D-FLER to erroneous inputs. From the implementation point of view, faster defuzzification methods are to be explored. [4] Finally, the topic of training and learning is an ample open research problem, especially complicated by the distributed nature of our fuzzy inference engine.

[4]See for example our later experiences with largest-of-maximum (LOM) and simplified centroid defuzzification in Chapters 6 and 7, respectively.

Chapter 6

Distributed Activity Recognition

Wireless sensor nodes can act as distributed detectors for recognizing activities online, with the final goal of assisting the users in their working environment. We propose an activity recognition architecture based on fuzzy logic, through which multiple nodes collaborate to produce a reliable recognition result from unreliable sensor data. As an extension to the regular fuzzy inference, we incorporate temporal order knowledge of the sequences of operations involved in the activities. The system outperforms non-fuzzy methods considered in previous work and can run online on sensor nodes, with execution times in the order of 40ms, for the whole recognition chain, and memory overhead in the order of 1.5kB RAM. This chapter represents a minor revision of the paper "Distributed Activity Recognition with Fuzzy-Enabled Wireless Sensor Networks" in International Conference on Distributed Computing in Sensor Systems (DCOSS), June 2008 [126], which is joint work with C. Lombriser, O. Amft, P. Havinga and G. Tröster. This work has been partially sponsored by the European Commission as part of the e-SENSE (contract no. 027227) and SENSEI (contract no. 215923) projects.

6.1 Introduction

Recent developments in wireless sensor networks and wearable computing [121] create the basis for augmenting the interaction between humans and highly-distributed, context-aware cognitive systems [104]. In particular, providing context-aware assistance to workers in industrial environments [30] has a number of potential benefits: (1) improves the productivity at work by supporting the workers with just-in-time, relevant information, (2) increases the reliability by supervising safety-critical process steps and (3) accelerates the learning curve of new, unskilled workers.

In this chapter, we focus on the concrete application of assembling and testing car body parts at a car manufacturing site [158, 39]. The mechanical assembly activities are supervised by a distributed system composed of wireless sensor nodes worn by the workers, embedded into the tools and deployed within the infrastructure. Having the technology integrated in the usual work environment ensures a simplified, unobtrusive human-machine interaction, which is important to foster user acceptance [81]. The same system can additionally support the training of new employees.

The most important technical objective of the system is *to recognize the user activities with high accuracy and in real time*, in order to provide prompt and exact assistance. We can expect several major difficulties related to: the inaccurate and noisy sensor data, the very limited resources (processing, memory and energy), the high variability of the data (e.g. for sensors placed on different parts of the body) and the unreliable, low bandwidth wireless communication.

To overcome these problems, we propose a distributed activity recognition system based on fuzzy-enabled WSNs. As already mentioned in Chapter 5, fuzzy logic represents nowadays a well-established field with an impressive number of successful applications in the industry and other connected areas [148]. It is our goal in this chapter *to explore the feasibility of distributed fuzzy inference for activity recognition and to compare its performance to state of the art methods.* With respect to our general methodology, based on prototyping and redesign, the study presented herein builds on and refers for comparison to the previous work of Amft et al. [39]. In our approach we decompose the activity recognition process in three steps: (1) the detection of action events using sensors on the body, tools, and the environment, (2) the combination of events into basic operations and (3) the final activity classification. For performance evaluation, we use the experimental data from 12 sensor nodes with 3D accelerometers, obtained during the car assembly trial conducted in [39] (see Figure 6.1).

The rest of this chapter is organized as follows. Section 6.3 presents the

overview of the distributed architecture. Section 6.4 analyzes the properties of the experimental data and derives the difficulties in achieving an accurate recognition. The detailed design of the FIS and the performance results are given in Section 6.5 and 6.6. As an extension to the normal fuzzy inference, we show how temporal order knowledge can improve the overall accuracy for the activities executed in sequences of operations. In order to study the feasibility of our solution, we implement the recognition algorithms on the Tmote Mini platform [27]. The results from Section 6.7 show that the algorithms can run online even on resource constrained sensor nodes. Section 6.8 discusses several important factors that can affect the performance of fuzzy-enabled WSN. Finally, Section 6.9 concludes the chapter.

6.2 Related Work

Activity recognition is a topic of high interest within the machine vision community. In particular, we can trace back the fundamental idea of dividing the recognition problem into multiple levels of complexity. In the "Inverse Hollywood Problem", Brand [52] uses coupled hidden Markov models (HMM) to visually detect causal events and fit them together into a coherent story of the ongoing action. Similarly, Ivanov and Bobick [96] address the recognition of visual activities by fusing the outputs of event detectors through a stochastic context-free grammar parsing mechanism. Wren et al. [181] show that even low resolution sensors (motion detectors and ultrasonic sensors) can provide useful contextual information in large buildings.

From a different perspective, distributed event detection has received considerable attention in the field of WSN. The research focuses on fundamental issues of WSN, such as reducing the number of messages needed for stable decisions [142], minimizing the effects of data quantization and message losses [150], and using error correcting codes to counteract data corruption [169]. The sensor nodes are typically assumed to sample at low data rates and to implement simple binary classifiers based on maximum likelihood estimates. However, recent work [155] shows that sensor nodes are able to execute more elaborate tasks, such as the classification of sound data. This result opens promising perspectives for using WSN in complex activity recognition processes.

The general problem of inferring complex events from simple events is also considered in middleware systems, such as DSWare [116] or Context Zones [138]. Still, there is a gap between the inference process in these systems and the algorithms operating with low-level sensor signals. In particular, it is difficult to

Figure 6.1: A car assembly worker performing several activities. Sensors are both worn by the worker and attached to the tools and car parts.

evaluate the influence of different levels on the overall recognition performance.

The field of wearable computing [121] represents a promising alternative for bridging this gap between high-level inference and low-lever sensor signal processing. The previous work of Amft et al. [39, 38] shows that inertial body-worn sensors can be used to extract relevant features from the continuous stream of data, process these features to detect atomic events and furthermore combine the events to eventually recognize complex human activities. This approach is demonstrated in the already mentioned car assembly trial, where the activity recognition is performed through two alternative methods: *edit distance* and *event histogram*. The edit distance approach relies on string matching, where strings are formed by assigning to each event type a unique character. The event histogram method compares the frequencies of event occurrences in specific activities.

The work we present in this chapter extends and complements the methods proposed by Amft et al. We propose an alternative approach based on fuzzy logic, specifically designed for running distributively and entirely on sensor nodes, as the final goal is to classify the activities online within the WSN. Our lightweight fuzzy inference engine can execute on the limited hardware of sensor nodes and proves more robust to inaccurate sensor data, achieving a better overall recognition performance than the edit distance and event histogram methods.

6.3 Activity Recognition Architecture

The analysis of the car assembly scenario resulted into the following architecture design issues to be considered:

1. Multiple *detectors* are available: sensors worn on the body, attached on the objects and tools, or deployed within the infrastructure. Although heterogeneous, most of these detectors have limited capabilities and low accuracy. In the considered car assembly experiment, the detectors are equipped with low-power, three-axial accelerometers.
2. Despite their limitations, the detectors can use simple methods to extract online the relevant features of the activities from the sampled data. Online processing is a design choice imposed by the low WSN bandwidth, which makes it unfeasible to transport large amounts of data to a central fusion point. Since the features are computed over a sliding time window and reported asynchronously by different detectors, we refer to them as *events*. For example, a detector worn on the right hand can spot the event "Pick-up something" based on a series of features identified in the continuous stream of acceleration data. For a detailed description and evaluation of event detectors, the reader is referred to [39].
3. A single event from one detector gives just an estimation of the real activity going on as it is perceived by the sensor at this location. However, fusing the information from several detectors reporting similar (or correlating) events within the same timeframe leads to a high confidence in recognizing that the user is performing a certain *basic operation*. For example, the basic operation "Mount the screws" can be inferred from the events signaled by the sensors on the user's arm, on the screwdriver and on the car part being assembled.
4. Sequences of basic operations form *complex activities*, which represent the final output of the recognition system. For example, the complex activity "Mount the brake light" consists of three basic operations: "Pick-up the brake light", "Insert the brake light" and "Mount the screws". We notice from this example that the order of the operations can be an important parameter for distinguishing among different activities (see Section 6.5.2 for details on how the execution order is used to improve the overall recognition performance).

Figure 6.2 summarizes the observations above. We denote the set of detectors by $\mathcal{D} = \{D_1, D_2, ..., D_d\}$. Each detector D_i can produce the set of events $\mathcal{E}_i = \{E_{i1}, E_{i2}, ..., E_{ie_i}\}$. Events from various detectors are combined to obtain the basic operations $\mathcal{O} = \{O_1, O_2, ..., O_o\}$. The identification of the basic operations represents the first level of data fusion. The aim is to reduce the number of inputs and consequently the classification complexity. The second level of data

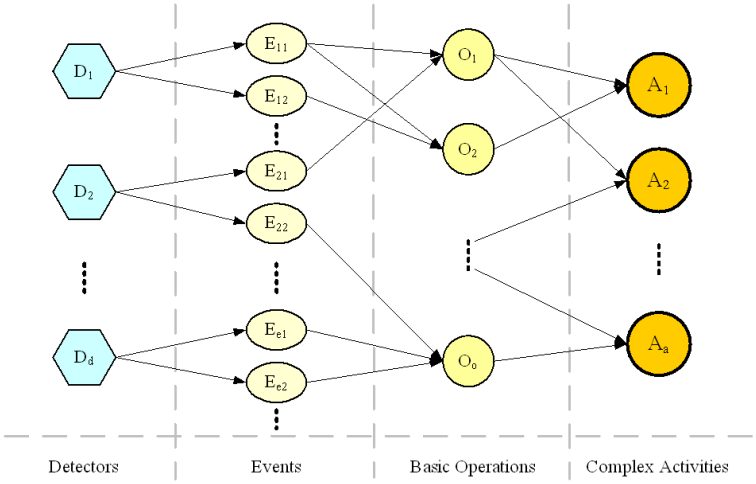


Figure 6.2: Distributed activity recognition architecture. The three main steps are: the detection of simple events (see also [39]), the combination of events into basic operations and the final activity classification.

fusion performs the combination of multiple observed basic operations into more complex activities $\mathcal{A} = \{A_1, A_2, ..., A_a\}$, where each A_i is composed of a set of operations $\{O_{i_1}, O_{i_2}, ..., O_{i_k}\}$, $1 \leq i \leq a$, $1 \leq k \leq o$.

There are two types of problems that adversely affect the performance of the recognition system. First, the low accuracy of the detectors may result into reporting false events (also referred to as *insertions*) or missing events (also referred to as *deletions*). Second, there are overlaps among different basic operations and also among different complex activities (i.e. the same events are involved in more operations and the same operations are performed during various activities), which can lead eventually to a misclassification due to confusion.

Fuzzy logic can be used throughout the whole activity recognition chain. The detectors usually identify the events with a certain confidence and cannot distinguish perfectly among similar events. This situation maps well to the notion of membership functions. Likewise, combining events into basic operations is similar to fuzzy majority voting, where different weights are assigned to the detectors based for example on their placement. Rule-based fuzzy inference is appropriate for the final activity classification, by checking the occurrence of the right operations. An interesting extension is to include the time sequence

Detector	Placement	Detector	Placement
D_1	Right arm	D_7	Trunk door
D_2	Left arm	D_8	Screwdriver 1
D_3	Upper back	D_9	Screwdriver 2
D_4	Front light	D_{10}	Front door
D_5	Brake light	D_{11}	Back door
D_6	Hood	D_{12}	Socket wrench

Table 6.1: Placement of the detectors.

Basic operation	Events (Detectors)
O_1-Pick-up the front door	$9(D_1)$, $15(D_2)$, $39(D_{10})$
O_2-Attach the front door	$40(D_{10})$
O_3-Mount the screws	$6(D_1)$, $20(D_3)$
O_4-Pick-up the socket wrench	$7(D_1)$, $21(D_3)$, $47(D_{12})$
O_5-Use the socket wrench	$10(D_1)$, $48(D_{12})$
O_6-Return the socket wrench	$49(D_{12})$
O_7-Close the front door	$16(D_2)$

Table 6.2: Activity A_1-"Mount the front door".

of the operations in the inference process (see Section 6.5.2).

6.4 Experimental Data

In order to evaluate the performance of our system, we utilize the experimental data obtained by Amft et al. [39] from a car assembly trial. Throughout the trial, 12 sensors with 3D accelerometers were used as detectors. Their placement is shown in Table 6.1. The detectors could spot a total number of 49 distinct events. We use the confidences of recognizing the events, ranging between $(0; 1]$, as inputs to our fuzzy-based recognition system. As explained in Section 6.3, we first combine the events corresponding to a certain basic operation, then perform the complex activity classification. The following example will best explain this process.

Let us consider the complex activity A_1-"Mount the front door". [1] Table 6.2 lists in the left column the operations involved in this activity, O_1 to O_7. The right column enumerates the events (represented as numbers) and the cor-

[1] The complete description of the activities A_2 to A_{11} can be found in Appendix 6.10.

responding detectors (given in brackets). For example, operation O_1-"Pick-up the front door" is inferred from events 9, 15 and 39 reported by detectors D_1, D_2 and D_{10}, respectively. We distinguish the following cases:

- *Operations identified by only one detector/event, such as* O_2. The confidence of executing such a basic operation is given by the associated detector/event confidence.
- *Operations identified by several detectors on the body, such as* O_3. Compared to the previous case, the combined confidence is higher, but still solely dependent on the accuracy of the gesture recognition. For computing the confidence of the basic operation, the average of the event confidences can be used. If multiple detectors are available, we can apply a fuzzy majority measure, as used in the D-FLER engine presented in Chapter 5.
- *Operations identified by several detectors on the body and on the tools (e.g.* O_4*) or the car parts (e.g.* O_1*).* This is the best case, as we can fuse information from different sources and produce a more reliable detection. The fusion can be done through a simple weighted average or a fuzzy majority measure, if more detectors of the same type are available.

Figure 6.3 depicts a grayscale map of the whole car assembly experiment. On the horizontal axis we have 11 complex activities, each of them performed 10 times by each of two subjects. The total recorded data amounts to approximately 5 hours. On the vertical axis we have the basic operations derived from the events reported by the detectors. By using a first level of data fusion, we reduced the number of input variables from 49 events to 31 basic operations. The gray spots indicate the confidence levels of the basic operations; higher confidences correspond to darker regions. The solid line rectangles mark the real operations involved in each of the 11 activities.

We observe that the fuzzy-based classification method has to cope with the following difficulties:

- Activities with significant overlapping definitions, such as A_1 and A_2, which have the operations O_3 to O_7 in common.
- Activities such as A_3 and A_4, which have theoretically only O_7 in common, but in practice record a large amount of false detections (insertions) causing overlaps.
- Constantly high-confidence insertions, for example O_{29} during A_1 and A_2.

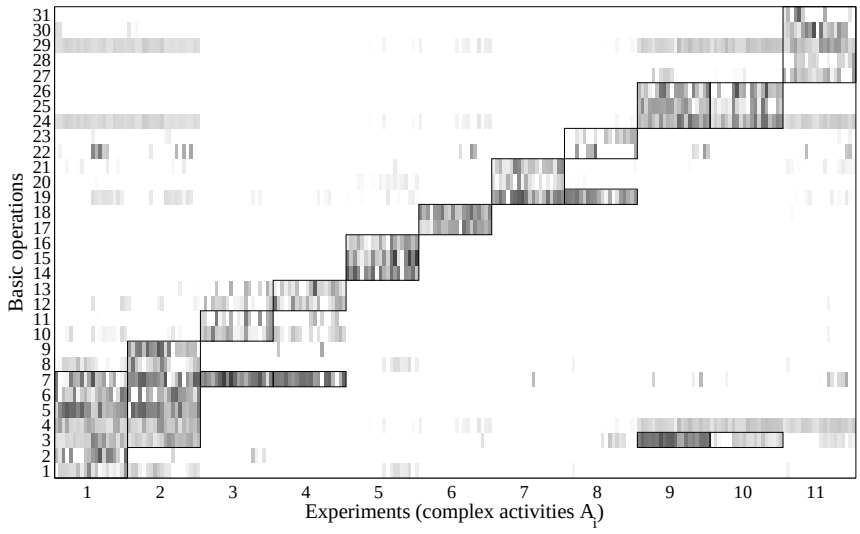


Figure 6.3: Basic operations observed by the detectors in the car assembly experiment. The gray spots represent confidence values, where a darker spot means a higher confidence. Operations relevant for the activity are marked by rectangles.

- Missed detections (deletions), such as O_{22} during A_8 or O_{31} during A_{11}. The continuity of the deletions in these examples suggests that they are caused by packet losses at the associated detectors.

6.5 Fuzzy-based Activity Recognition

In this section we present in detail the design the fuzzy-based recognition system. We first analyze several alternatives for the main FIS components, as well as for training and learning, then extend the inference for sequences of operations ordered in time.

6.5.1 Fuzzy System Components

The experimental data described in Section 6.4 is used as input to the fuzzy inference for the final activity classification. There are three important components of the FIS: the membership functions, the rule base and the defuzzifier. Making the right choices for these components is the most difficult task in fuzzy system design. However, the difficulties can be leveraged if (1) there is expert knowledge about the process to be modeled and/or (2) there is a collection of input/output data that can be used for learning and training. The car assembly scenario matches both conditions. Therefore, we base the FIS design on the following list of facts derived from the actual process:

1. The number of output classes is known, equal to the number of complex activities (11).
2. The mapping of inputs (basic operations) to outputs is known from the definition of each activity (see Appendix 6.10).
3. The input data is noisy and contains insertions (see Section 6.4) that generate confusion among output classes.
4. The erroneous wireless communication causes deletions in the input data (see Section 6.4), which translate into non-activation of the fuzzy rules.

Fact 2 allows us to define the rule base. Each complex activity is selected by a rule taking the inputs corresponding to the activity definition. For example, activity A_7 (see Appendix 6.10) generates the rule:

$$\begin{aligned} \text{IF} \quad & O_{19} \text{ is } High \text{ AND } O_{20} \text{ is } High \text{ AND } O_{21} \text{ is } High \\ \text{THEN} \quad & A_7 \text{ is } High \end{aligned} \tag{6.1}$$

The next step is to choose the membership functions for the fuzzy sets. In the following, we discuss three alternatives: two well-established automatic learning techniques and a simple heuristic method.

1. *Fuzzy c-means clustering (FCM)* [47]. Fact 1 indicates FCM as a possible algorithm for learning the membership functions, since we know beforehand the number of clusters. However, the FIS generated by FCM has two major drawbacks in our case. First, FCM generates Gaussian membership functions centered around the coordinates of the cluster centers. This does not conform to our classification model, where a higher input value (i.e. confidence closer to 1) gives more certainty that the output

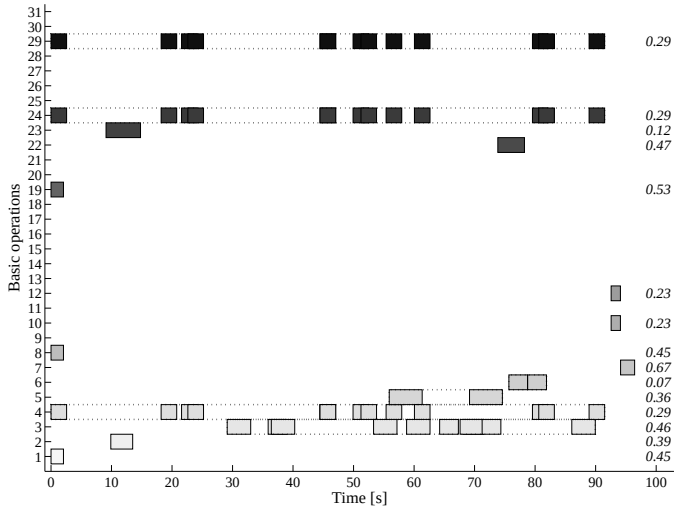


Figure 6.4: Basic operations reported during activity $A_1 = \{O_1, \ldots, O_7\}$. The operations in the upper part are mistakenly inserted by the detectors. Multiple occurrences of operations are combined (dotted lines) to analyze the correct order in time.

belongs to a particular class. Second, FCM increases the computational overhead because (1) it uses Gaussian membership functions and (2) it takes into account all the inputs in each rule, while we have partial rules, such as the rule from Equation 6.1.

2. *Adaptive Neuro-Fuzzy Inference System (ANFIS)* [98]. ANFIS generates a FIS based on a given input/output data set. The membership function parameters are tuned using a hybrid learning method that combines gradient descent and least-squares. For partitioning the input space, ANFIS uses grid partitioning or subtractive clustering. Because we have a large number of inputs (31), grid partitioning is not an appropriate method, as it leads to an exponential explosion of the number of rules, known as the "curse of dimensionality" problem. Therefore, we run ANFIS with subtractive clustering on our experimental data set. We obtain unacceptably large training and checking errors, which indicate that the collected data does not capture all the features needed for optimal training with ANFIS.
3. *Simple heuristic method.* For computational simplicity, we use trapezoidal

membership functions. The first problem is to choose the upper threshold of the membership functions. For each input O_i, we compute the threshold as the mean value $m(O_i)$ over the training set. Consequently, confidences higher than $m(O_i)$ will be fuzzified to 1. The second problem arises from Fact 4: deletions (i.e. zero value inputs) determine the non-activation of the appropriate rules when using the usual *max-min* or *sum-product* inference. To alleviate this effect, we lower the zero threshold of the membership functions, so that zero input values are fuzzified to strictly positive numbers. The downside of this approach is that the level of confusion among output classes increases. We will see however in Section 6.6 that this has no major impact on the overall performance.

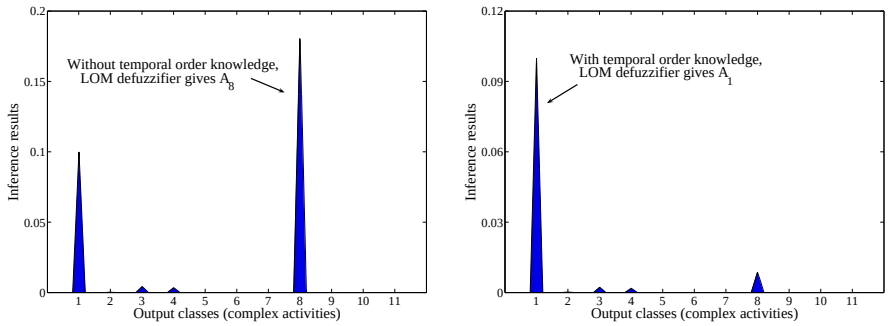
The last component to discuss is the defuzzifier. The standard centroid-based defuzzification is the most time-consuming operation on a sensor node, taking up to 95% of the total inference time [124]. For the activity classification problem, however, Fact 3 suggests that the *largest-of-maximum* (LOM) defuzzification method is more appropriate because it favours the most probable activity to be selected at the output. In addition, as shown in Section 6.7, LOM defuzzification has a much faster execution time than the centroid method on sensor nodes.

6.5.2 Temporal Order of Basic Operations

In this section, we extend the recognition system for the case of sequences of ordered operations by incorporating time knowledge in the fuzzy inference process. To illustrate the usefulness of this extension, we return to the example of activity A_1 from Table 6.2.

Figure 6.4 shows one experimental data set collected when performing A_1 during the car assembly trial. The basic operations detected are represented as gray-tone rectangles. The position of the rectangles on the vertical axis gives the basic operation ID. The width of the rectangles gives the extent of time during which the operations were reported by the corresponding detectors. On the right side of the figure, we list the average confidence of each operation. We remember from Table 6.2 that activity A_1 consists of operations $O_1, O_2, ..., O_7$, in this order. Analyzing Figure 6.4, we can make the following observations:

- The required operations $O_1, O_2, ..., O_7$ are all recognized. However, O_6 has very low confidence (0.07).
- There are recurring insertions, for example O_{29}, with confidence 0.29 on the average. Nevertheless, this does not generate a classification error



(a) Without temporal order (b) With temporal order

Figure 6.5: (a) The insertions observed in Figure 6.4 determine the incorrect result A_8 instead of A_1. (b) Using the penalty function T_A for operations appearing in the wrong time order, A_1 is correctly identified.

because O_{29} is involved only in A_{11}, together with $O_{27}, O_{28}, O_{30}, O_{31}$, and none of these operations occurs during the experiment.

- There are also insertions of O_{19}, O_{22}, O_{23}, which form the activity A_8- "Mount hood rod". O_{19} and O_{22} have high confidences (0.47 and 0.53). Even though O_{23} has a lower confidence (0.12), this is fuzzified to a high value because O_{23} occurs with low confidence throughout the whole trial (so also in the training set).

As shown in Figure 6.5 (a), the result of the LOM defuzzification in this case is wrong; the FIS classifies the activity as A_8 instead of A_1.

Time knowledge can help overcome these problems. Let us analyze again Figure 6.4. First, we build the unified time intervals (represented as dotted line rectangles) when each operation was recognized. Then, we define the strict temporal order relation $\prec$ between two operations O_i and O_j with their time intervals $T_i = [a_i; b_i]$ and $T_j = [a_j; b_j]$, respectively, as:

$$O_i \prec O_j \iff b_i < a_j \tag{6.2}$$

The $\prec$ relation is used to identify the operations appearing in wrong order, i.e. the cases when O_i should be executed *after* O_j according to the activity description, but $O_i \prec O_j$ in the experimental data. Figure 6.6 shows several examples of time intervals for operations O_i and O_j. In the rightmost situation

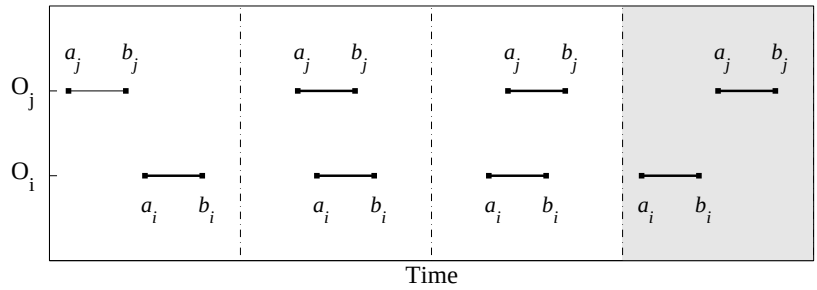


Figure 6.6: Examples of time intervals for operations O_i and O_j. If O_i should be executed after O_j according to the activity description, the rightmost situation (highlighted) is wrong and is considered an inversion.

(highlighted) $O_i \prec O_j$, which is wrong and is considered an inversion. We notice that when the unified time intervals T_i and T_j overlap, the operations are *not* considered in the wrong order, so $\prec$ is a strict relation. Looking back at Figure 6.4, operations $O_1, O_2, ..., O_7$ appear in the right order according to the definition of A_1. In contrast, $O_{19} \prec O_{23} \prec O_{22}$, while the correct sequence for A_8 would be O_{22}, O_{19}, O_{23}. If we filter out such incorrectly ordered insertions, we can prevent errors as in Figure 6.5 (a). For this purpose, we need to add to the initial FIS an input variable for each activity class; these variables characterize how well the sequences of operations fit the activity descriptions.

Without the loss of generality, let us consider activity A defined as $A = \{O_1, O_2, ..., O_k\}$ and the maximal sequence of operations from experimental data $S = \{O_{i_1}, O_{i_2}, ..., O_{i_l}\}$, with $S \subseteq A$. [2] We define the number of inversions in S with respect to A (see also Figure 6.6) as:

$$inv_A(S) = \|\{(i_a; i_b) \mid a < b \text{ and } O_{i_b} \prec O_{i_a}\}\| \tag{6.3}$$

and the number of deletions:

$$del_A(S) = \|A\| - \|S\| = k - l \tag{6.4}$$

where $\|.\|$ is the cardinality measure.

In other words, $inv_A(S)$ measures the number of pairs of operations in the wrong order and $del_A(S)$ the number of deletions with respect to A. In the ideal

[2] Insertions of operations not related to A are not taken into account in the input variable for A (T_A), but in the input variables for the other activities, according to their definitions.

case, S contains the same operations as A and in the same order, meaning that $inv_A(S) = del_A(S) = 0$.

We use the measure:

$$T_A = \frac{inv_A(S) + del_A(S)}{k(k+1)/2} \tag{6.5}$$

as an input variable to the FIS for each activity A, where S is the maximal sequence included in A from the experimental data. By including both the inversions and deletions, T_A acts as a penalty function for matching the sequence of detected operations to the right activity description. For the example in Figure 6.4 and activities A_1 and A_8, we have $T_1 = (0+0)/28 = 0$ and $T_8 = (2+0)/6 = 0.33$, respectively, so T_8 will induce a higher penalty.

The rule base is also extended to take the new inputs into account. For example, the rule from Equation 6.1 (see Section 6.5.1), corresponding to activity A_7, becomes:

$$\begin{aligned} \text{IF} \quad & O_{19} \text{ is } High \text{ AND } O_{20} \text{ is } High \text{ AND } O_{21} \text{ is } High \text{ AND } T_7 \text{ is } Low \\ \text{THEN} \quad & A_7 \text{ is } High \end{aligned} \tag{6.6}$$

The result of the fuzzy inference taking into account the new T_A variables is depicted in Figure 6.5 (b). As we can see, the LOM defuzzification yields the correct classification result, A_1.

6.6 Results

As explained in Section 6.4, the car assembly trial consisted of 11 complex activities performed 10 times by each of the 2 subjects, so a total of 220 experiments. In this section, we present the performance results of both the normal FIS and the temporal order extension. As a basis for comparison, we refer to the previous work of Amft et al. [39], where the same car assembly trial was analyzed with two methods: *edit distance* and *event histogram*.

The activity recognition is performed by processing the events from detectors through an adaptive search window (see also [39]). The system slides this search window over the continuous flow of events and feeds the events contained in the search window into the fuzzy inference engine. The inference result is compared to an activation threshold in order to decide whether there is really an activity going on or the current situation should be classified as "nothing happens" (NULL class). This procedure influences the recognition performance

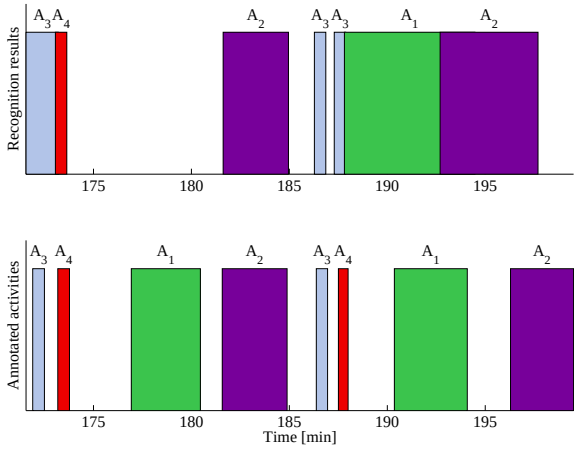


Figure 6.7: Snapshot of the recognition results. The upper graph depicts the recognized activities and the lower graph shows the ground truth.

in the following ways. Firstly, the size of the search window may not match the actual duration of the complex activity and, consequently, relevant events are left out or additional events are taken in the inference process. This can result into a wrong classification. To deal with this problem, the size of the search window is adapted according to the selected set of training data. Secondly, the activation threshold affects directly the accuracy of disambiguation between the NULL class and actual activities. This section will present later on how the threshold can be tuned to find the optimal balance between false positives and false negatives.

To illustrate these problems, Figure 6.7 gives a snapshot of the continuous recognition process. The upper graph depicts the activities reported by the fuzzy inference and the lower graph represents the annotated data (ground truth). We notice that activity A_1 is not recognized at time 179 (i.e. considered to be NULL class) and A_4 is confused with A_3 at time 188. The remaining activities are correctly recognized. Since we only report the search window in which each activity occurred, the recognized time intervals are not perfectly aligned to the ground truth, but just overlapping (especially visible in the case of A_1 and A_2 at times 191 and 196, respectively).

In order to characterize thoroughly the recognition performance, we analyze the metrics *recall* and *precision*. Traditionally used in information retrieval

systems, these metrics are defined as follows [53]:

- *Recall* is the number of retrieved relevant items as a proportion of all relevant items. In the case of activity recognition, recall can be viewed as a measure of effectiveness in identifying and classifying the activities. However, high recall is not enough to have an accurate recognition system, as it does not account for possible false positives. Formally, recall is computed as:

$$Recall = \frac{Retrieved \cap Relevant}{Relevant} \tag{6.7}$$

- *Precision* is the number of retrieved relevant items as a proportion of the number of retrieved items. Precision is, therefore, a measure of purity in retrieval performance, and, in activity recognition, a measure of effectiveness in excluding nonrelevant situations (most commonly, the NULL class) from the reported results. Formally, precision is computed as:

$$Precision = \frac{Retrieved \cap Relevant}{Retrieved} \tag{6.8}$$

Figure 6.8 shows the recall and precision of the fuzzy-based system, using four-fold cross validation [3] for the selection of training and validation data (the training of the membership functions follows the heuristic method described in Section 6.5.1). The normal fuzzy inference achieves an overall recall and precision of 0.81 and 0.79. The temporal order extension improves the recognition performance to 0.85 recall and 0.85 precision. Figure 6.9 compares these results to the edit distance (0.77 recall and 0.26 precision) and event histogram (0.77 recall and 0.79 precision) methods from [39]. We make the following observations:

1. Activities A_6 and A_7 display a recall and precision close to 1, while A_4 and A_{10} have a relatively low recognition performance. This behavior can be explained by analyzing again Figure 6.3. We notice the clear separation in the input data for A_6 and A_7, which leads to the high recognition performance. Likewise, we notice the similitude between the input data for A_3 - A_4, which generates the confusion in the recognition results.

[3] In K-fold cross-validation, the data sample is partitioned into K subsamples. $K-1$ subsamples are used as training data and the remaining subsample is used as the validation (test) data. The process is repeated K times, with each of the K subsamples being retained once as the validation data. The final results represents the average over the K tests.

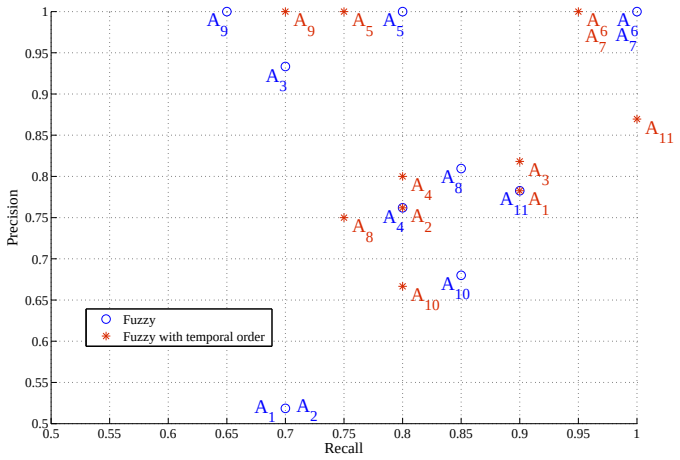


Figure 6.8: Recognition performance results of both the normal FIS and the temporal order extension. The flow of events from the detectors is processed online through an adaptive search window. The temporal order extension improves considerably the recognition performance of complex sequences of operations (A_1, A_2, A_{11}), but may be more sensitive to deletions in the case of simple activities (A_8).

2. The normal FIS has low performance in the case of A_1 and A_2, which are the most complex activities, each comprising 7 basic operations. The temporal order extension improves considerably the recognition performance of these activities, increasing the recall and precision by 15% and 25% on average. Similarly, the recall and precision of A_{11} (composed of 5 operations) are increased by 10% and 9%, respectively. This shows that analyzing the temporal order is particularly successful in the case of complex sequences of operations.

3. For less complex activities, such as A_8 (composed of only 3 operations), the performance of the temporal order extension may drop compared to the regular fuzzy inference. This is caused by the penalty for deletions introduced via the $del_A(S)$ factor (see Eq. 6.4), which becomes considerable in the case of a small k (number of operations).

4. The normal fuzzy method outperforms the event distance in terms of precision and has similar performance as the event histogram, with a slight

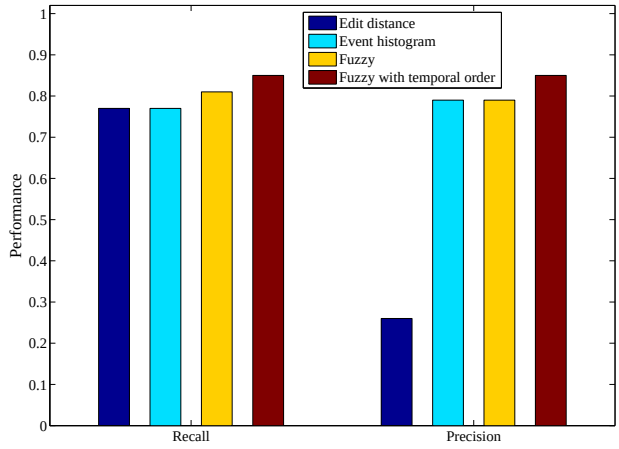


Figure 6.9: Comparison of the overall recall and precision achieved by four recognition methods: edit distance, event histogram [39], normal fuzzy and the temporal order extension.

improvement in the recall. The temporal order extension further increases the recognition performance with 4-6% on average. Therefore, the event histogram and normal fuzzy methods are appropriate for activities composed of few basic operations, while the temporal order extension is recommended for more complex activities.

As a final result, we present the tradeoff between the overall recall and precision in Figure 6.10. It is known from information retrieval theory that a tradeoff between recall and precision is unavoidable [53]. In our case, the tradeoff is given by the activation threshold set on the FIS output for disambiguating the nonrelevant situations from real activities. More specifically, if the FIS output is lower than the threshold, we consider the situation as "nothing happening" (the NULL class), otherwise we report the classified activity A_1-A_{11}. As depicted in Figure 6.10, the threshold value creates the typical inverse relation between recall and precision, for both the normal FIS and the temporal order extension. This result is of particular importance because it shows how the recognition performance can be adapted according to what is more important from the application perspective: to have a low rate of false positives (good precision) or to minimize the false negatives (good recall).

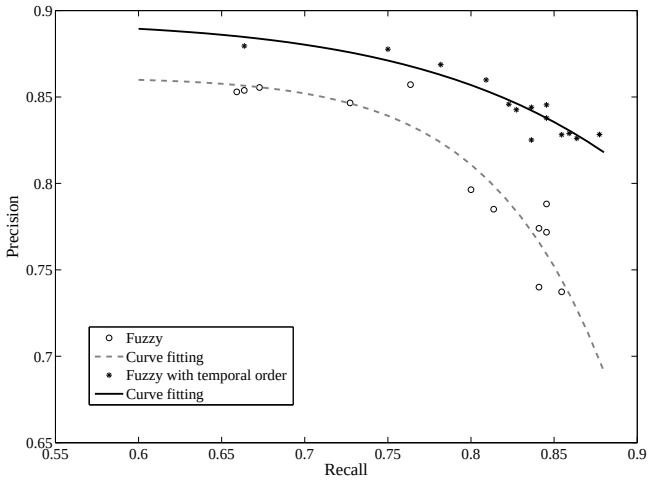


Figure 6.10: Tuning the performance of the fuzzy-based methods. The tradeoff results from the threshold value used in filtering the NULL class at the FIS output.

6.7 Prototyping

In order to analyze the feasibility of our approach, we implemented the detectors algorithms and the fuzzy inference on a resource-constrained WSN platform. We chose the Tmote Mini sensor node platform [27] (see Figure 6.11 (a)) for its compact, industry-standard miniSDIO form factor. The Tmote Mini integrates an MSP430 microcontroller (8MHz, 48kB FLASH, 10kB RAM) and a TI/Chipcon CC2420 low-power radio (2.4GHz IEEE 802.15.4-compliant) in a single one square inch package.

The implementation of the detectors covers two main tasks: (1) to spot relevant sensor data segments in a continuous data flow and (2) to classify those segments as one of the detector's events. To this end, the detectors implement a feature similarity search (FSS) algorithm [38, 39]. In the first phase, the data is segmented into windows of a detector-dependent size with a step size of 250ms. In the second phase, the algorithm extracts a detector-dependent number of features from the data segments, and forms the feature vector. The similarity (measured by the Euclidean distance) of this feature vector to a set learned from user-annotated data determines the classification decision.

Let us choose detector D_{10} (front door) for example. D_{10} implements 10

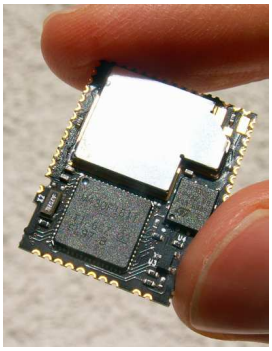
features (length, beginning, end, total difference from mean on 5 partial windows, variance) from acceleration data sampled at 50Hz using an ADXL330 3-axis accelerometer. The complexity of the FSS algorithm scales with the window length (200 samples), the number of features and the number of classes to be distinguished. The total execution time measured on the sensor node is 28.8ms, more precisely 26.8ms for the feature computation phase and 2.0 ms for the classification phase. The algorithm requires 1.3kB of RAM and 1.9kB of FLASH.

The implementation of the fuzzy inference covers the main components described in Section 6.5:

- *Fuzzification.* For computational simplicity, we use trapezoidal membership functions. The fuzzification is computational oriented due to the limited RAM available. To reduce the computational overhead, the maximum fuzzified value is scaled to a power of 2, as recommended by Dannenberg [67].
- *Inference.* The rules base is stored in a compact, binary form. We use *sum-product* inference as it proves more robust to deletions than max-min method.
- *Defuzzification.* The result of the rule evaluation is defuzzified using the *largest-of-maximum* (LOM) method.

Figure 6.11 (b) presents the performance parameters of our implementation, for both the normal FIS and the temporal order extension. We can conclude that a total execution time of less than 12ms is feasible on the MSP430 microcontroller. Moreover, the defuzzification time is almost negligible compared to fuzzification and inference. This result highlights the considerable benefit of using LOM instead of centroid defuzzification (for comparison, see [124]). The rightmost column of Figure 6.11 (b) lists the amount of RAM required by the FIS, out of 10kB available. The code memory footprint amounts to 490 bytes, out of 48kB available. These figures confirm that a very lightweight implementation of the FIS is feasible.

A final important aspect is the numerical accuracy of the FIS running on the node. In order to quantify the impact of integer computation over the fuzzy inference, we input the experimental data sequentially to the sensor node via the serial port and collect the classification results. Similarly to our results with D-FLER from Chapter 5, the average error compared to floating point computation is 1.11%. Due to the impracticality of floating point computation on the node, we consider this value within reasonable tolerance limits.



(a)

	Normal	Time order
No. of inputs	31	42
RAM [bytes]	288	344
Fuzzification [ms]	4.21	5.88
Inference [ms]	4.45	5.97
Defuzzification [ms]	0.07	0.07
Total time [ms]	8.73	11.92

(b)

Figure 6.11: (a) Tmote Mini sensor prototype platform. (b) FIS implementation details: execution time and memory footprint.

6.8 Discussion

The results presented so far give an evaluation of the activity recognition system for a particular experiment and the associated data set. In what follows, we provide a more general discussion of the relevant performance factors, with respect to the particularities of fuzzy-enabled WSN.

Distributed recognition architecture. A distributed architecture is required when implementing activity recognition on resource-constrained devices because (1) a powerful central node may not be available and (2) the WSN does not have the capacity of transporting the raw data to a central fusion point. However, as we showed in the previous sections, it is essential that the distributed recognition system considers the whole chain from sensor data to activity classification, with respect to both performance evaluation and implementation.

Recognition performance. Fuzzy logic proves to be robust to both unreliable sensor data and wireless link failures, which cause insertions and deletions. The analysis of the temporal order improves the recognition performance, especially for the activities with many operations executed in a clear sequence. However, the performance remains limited for the activities that are either very similar or have large overlapping areas in the input data. Two additional factors can further affect the overall performance. The first concerns filtering the NULL class, which makes a tradeoff between precision and recall. The second

is related to the segmentation problem: the recognition system running online cannot know the exact extent of the activities, so only the events within a certain time segment are detected. Consequently, the FIS may operate with incomplete inputs and yield an incorrect result.

Training and learning. The expert knowledge is of foremost importance in designing the FIS, in particular the rule base. For optimal performance, further tuning of the membership functions may be needed. As mentioned in Section 6.5.1, the FCM algorithm represents a useful tool for automatic generation of membership functions. Several post-processing steps are still necessary, for example converting to trapezoidal functions and removing the unnecessary terms from the rule antecedents.

Energy consumption. There are two main factors that influence the energy consumption. The first factor is the power dissipated on the actual sensors (e.g. accelerometers), which is independent of the activity recognition system. The second important factor is the wireless communication. The distributed recognition architecture keeps the network overhead to a minimum, by communicating only the spotted events. The number of messages sent by a detector node D_i is proportional to: (1) the amount of events D_i can detect, i.e. $\|\mathcal{E}_i\|$, (2) the frequency of occurring of each event E_{ij} and (3) the rate of insertions. The number of messages received by a node depends on its role in the data fusion process, which can be: (1) simple detector, (2) combining events into basic operations or (3) final activity classifier. It follows that the latter two roles should be assigned to nodes that have fewer (or none) events to detect.

Memory and computational overhead. The implementation on sensor nodes shows that the memory requirements are very low and both the event detection algorithms the fuzzy inference achieve execution time of approximately 40ms. Although the major concern in WSN is to keep the communication overhead to a minimum, fast execution times are equally important in our case, because three demanding tasks compete for CPU utilization: the sensor sampling, the medium access protocol and the activity recognition.

6.9 Conclusions

Wireless sensor nodes worn on the body and placed on the tools can recognize the activities of the users and assist them at work. We presented a distributed architecture that uses fuzzy logic for reliably detecting and classifying the user activities in real-time. For performance evaluation we used experimental data from 12 sensor nodes equipped with 3D accelerometers, obtained during a car

assembly trial within an industrial setting. The fuzzy system achieved an overall recall and precision of 0.81 and 0.79. An interesting extension was to include temporal order knowledge about the sequences of operations into the fuzzy inference, which improved the recognition performance to 0.85 recall and 0.85 precision. In order to analyze the feasibility of our approach, we implemented both the detection algorithms and the fuzzy logic engine on the Tmote Mini sensor platform. The results showed that our method can run in real-time, with execution times of approximately 40ms, memory overhead in the order of 1.5kB and an overal accuracy loss of 1.11%.

6.10 Appendix

The description of the complex activities A_2 to A_{11}, decomposed into basic operations and the corresponding events/detectors.

Activity	**Basic operation**	**Events (Detectors)**
A_2-Mount the back door	O_8-Pick-up the back door	9(D_1), 15(D_2), 43(D_{11})
	O_9-Attach the back door	44(D_{11})
	O_3-Mount the screws	6(D_1), 20(D_3)
	O_4-Pick-up the socket wrench	7(D_1), 21(D_3), 47(D_{12})
	O_5-Use the socket wrench	10(D_1), 48(D_{12})
	O_6-Return the socket wrench	49(D_{12})
	O_7-Close the back door	16(D_2)
A_3-Test the front door	O_{10}-Open the front door	17(D_2), 41(D_{10})
	O_{11}-Teeter the front door	18(D_2), 42(D_{10})
	O_7-Close the front door	16(D_2)
A_4-Test the back door	O_{12}-Open the back door	17(D_2), 45(D_{11})
	O_{13}-Teeter the back door	18(D_2), 46(D_{11})
	O_7-Close the back door	16(D_2)
A_5-Test the trunk door	O_{14}-Open the trunk door	30(D_7)
	O_{15}-Teeter the trunk door	31(D_7)
	O_{16}-Close the trunk door	32(D_7)
A_6-Mount the brake light	O_{17}-Pick-up the brake light	25(D_5)
	O_{18}-Insert the brake light	19(D_2), 26(D_5)
A_7-Test the hood	O_{19}-Open the hood	1(D_1), 11(D_2), 27(D_6)
	O_{20}-Teeter the hood	2(D_1), 12(D_2), 28(D_6)
	O_{21}-Close the hood	3(D_1), 13(D_2), 29(D_6)
A_8-Mount the hood rod	O_{22}-Pick-up the rod	4(D_1)
	O_{19}-Open the hood	1(D_1), 11(D_2), 27(D_6)
	O_{23}-Install the rod	5(D_1), 14(D_2)
A_9-Mount the water tank	O_3-Mount the screws	6(D_1), 20(D_3)
	O_{24}-Pick-up the screwdriver	7(D_1), 21(D_3), 36(D_9)
	O_{25}-Use the screwdriver	37(D_9)
	O_{26}-Return the screwdriver	38(D_9)
A_{10}-Mount the bar	O_3-Mount the screws	6(D_1), 20(D_3)
	O_{24}-Pick-up the screwdriver	7(D_1), 21(D_3), 36(D_9)
	O_{25}-Use the screwdriver	37(D_9)
	O_{26}-Return the screwdriver	38(D_9)
A_{11}-Mount the front light	O_{27}-Pick-up the front light	23(D_4)
	O_{28}-Install the front light	24(D_4)
	O_{29}-Pick-up the screwdriver	7(D_1), 21(D_3), 33(D_8)
	O_{30}-Use the screwdriver	22(D_3), 34(D_8)
	O_{31}-Return the screwdriver	35(D_8)

Chapter 7

Mobile Sensor Team Coordination

This chapter explores the idea of mobile team coordination using devices with sensing, communication and actuation capabilities. Such collaborative teams of robots can improve productivity in process manufacturing and, in more visionary scenarios, enable cooperative surveillance, automated vehicular convoys and even perform space exploration missions. As a proof of concept, we use vehicles on wheels, augmented with wireless, sensing and control capabilities. The final goal is to have a self-organizing team (or swarm) of nodes that maintain a formation by periodically exchanging their sensed movement information. Each node features low-power inertial sensors, from which it computes speed and orientation online. The leader vehicle periodically transmits these measures to the followers, which implement a lightweight fuzzy logic controller for imitating the leader's movement pattern. The solution is not restricted to vehicles on wheels, but supports any moving entities capable of determining their velocity and heading, thus opening promising perspectives for machine-to-machine and human-to-machine spontaneous interactions in the field. This work is therefore exploratory on the use of WSN technology - characterized by significant low bandwidth, low power consumption and unreliable wireless communication - in the well-established fields of swarm robotics and industrial robotics.

7.1 Introduction

We have seen so far distinct facets of collaborative WSNs at several levels: networking, event detection, distributed reasoning. It is now time to make a step forward toward dynamic WSNs that can not only sense and process information, but also actuate and change the state of their environment in a coordinated fashion. With respect to industrial applications, this a more visionary scenario, where human operators, robots and WSNs form a holistic cognitive system that is able to adapt to open-ended and frequently changing real-world environments. By applying coordinated sensing, communication and actuation, the overall process can benefit from improved productivity, efficiency, on-site safety and enhanced man-machine interaction.

In this general frame, the current chapter addresses the specific problem of distributed movement coordination of vehicles equipped with wireless sensor nodes. The final goal is to have a self-organizing team (or swarm) of nodes that maintain a formation by periodically exchanging their sensed movement information. It is our explicit goal to provide a *fully localized* solution, without any external PC-based control, and based solely on low-cost, low-power inertial sensors (no cameras or GPS, low cost of the hardware platform). A broad range of application domains would benefit from having many such inexpensive wireless vehicles that autonomously team up and coordinate their movement, for example: cooperative surveillance, exploring unknown areas [55], disaster management [2], automated highway [85], truck convoys [79], space exploration [80]. This chapter is therefore exploratory on the use of WSN technology in the fields of swarm robotics and industrial robotics.

With respect to our general methodology, the work described herein is based on the prototype WSN system for movement-based group awareness presented in our previous studies [129]. We showed that sensor nodes equipped with tilt switches or MEMS accelerometers can autonomously determine that they move together, by communicating and correlating online their movement information. Starting from these results, we explore in this chapter the idea of inertial movement coordination based on the distributed sensor-actuator paradigm.

Mobile team coordination in WSNs faces a number of challenges. Firstly, executing all the tasks on the node – sensor sampling, processing, communication and control – may easily exceed the computational and memory resources available. Consequently, we must find the right *scheduling* that trades-off between accuracy and responsiveness, on the one hand, and sampling frequency and wireless communication duty cycle, on the other hand. Secondly, actuator nodes must run a navigation control loop for regulating the movement of vehi-

cles. Designing and implementing a suitable *controller* for this purpose is far from trivial on limited hardware. Thirdly, using inexpensive, low-power inertial sensors also means a relatively low accuracy and robustness to noise. Therefore, *calibration*, *filtering* and *dynamic error compensation* are strongly required for improving the quality of measurements.

In view of these challenges, the key contributions of this chapter are as follows. Firstly, we devise a miniaturized, low-cost navigation system using low-power wireless sensor nodes equipped with three-axial accelerometers and magnetic compasses. Secondly, we explore fuzzy logic as a lightweight and robust control solution for coordinating the group movement in a leader-follower fashion. Thirdly, we report on all development phases, covering simulation, controller tuning, inertial sensor evaluation, calibration, scheduling, fixed-point computation, debugging and benchmarking. Finally, we evaluate the performance of our prototype system through field experiments and we discuss the most important results.

The rest of this chapter is organized as follows. After overviewing the related work in Section 7.2, we describe the design details of our system: inertial navigation in Section 7.3, wireless communication in Section 7.4 and fuzzy control in Section 7.5. Section 7.6 presents our simulation framework. The implementation details are given in Section 7.7. Thorough experimental results are provided in Section 7.8. Finally, Section 7.9 concludes the chapter.

7.2 Related Work

The field of robotics accounts for extensive related work on swarms (or flocks) of robots [45]. The common objective is to achieve formation control and make multiple robots move collectively. Various technologies are used to determine inter-robot distances and orientations for the purpose of maintaining a formation: modulated infrared light [143], acoustic signals among submersibles [105], omnidirectional cameras and physical attachments [131]. Such robots typically feature powerful computing boards with clock frequencies ranging from 40 to 400 MHz. In comparison, our solution targets a more loosely-coupled coordination among resource-constrained devices, using only inertial sensors and low-power wireless communication.

In recent work, Allred *et al.* [37] describe SensorFlock, which is composed of small bird-sized nodes called micro-air vehicles (MAVs). The MAVs are expected to communicate wirelessly and follow a certain trajectory mission plan. The flight control system fuses information from the GPS and gyroscope sensors

on board. SensorFlock focuses on keeping the nodes autonomously in the air and provides an in-depth study of the RF characteristics and networking connectivity. In comparison, we focus on inertial sensing and explore fuzzy logic as a lightweight yet robust control method for coordinating the movements of mobile nodes in the field.

Wang *et al.* [171] overview several mobile WSN platforms based on MICA motes and running TinyOS, such as: MASmote, MICAmote, CotsBots and Robomote. The MAS-net project also considers the usage of MASmotes for formation control [170], including a leader-follower strategy. The MASmotes rely on a pseudo-GPS location system (based on cameras) and odometry to measure node displacement. In contrast, our solution is fully localized and does not require any infrastructure.

Fuzzy control for autonomous vehicles has also been considered by previous research. Kodagoda *et al.* [103] present an autonomous golf car controlled by a 450 Mhz PC. Simulations of similar systems are provided in [87] and [62]. Compared to these approaches, our system is much more resource constrained, exploits wireless communication and assumes a simpler physical vehicle model.

7.3 Navigation

As depicted in Figure 7.1, we consider the scenario of a *leader* vehicle, whose trajectory has to be copied by *follower* vehicles. The result is a moving ensemble that can be controlled from a single point or can follow a unique mission plan deployed only on the leader. Theoretically, synchronous movement is achieved when all the vehicles maintain the same *velocity* and *heading* with respect to a reference system. When moving, the leader computes its velocity and heading from sensor measurements, and broadcasts the data periodically to the followers. Each follower determines its own speed and heading, compares them with the information received from the leader and takes the necessary action to correct its trajectory if necessary. [1]

[1] Since we are not using any external reference (e.g. GPS) or relative distance sensors between the follower and leader, this method is limited with respect to convoy-like applications. Due to the accumulation of sensing errors, the distance between vehicles may change in time and, in worst cases, vehicles may collide or become too remote to maintain radio contact.

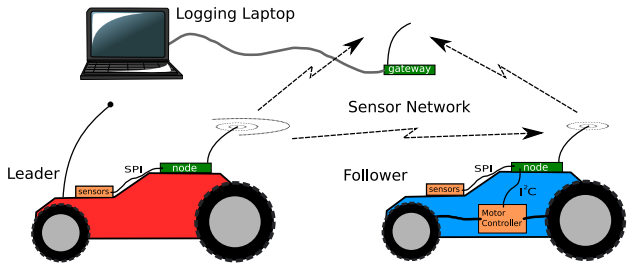


Figure 7.1: Overview of the leader-follower scenario.

7.3.1 Velocity Integration

In strapdown inertial navigation systems, velocity v is computed from the following equation [161]:

$$\frac{d\vec{v}}{dt} = \vec{a} - \vec{\omega} \times \vec{v} - \vec{\omega} \times (\vec{\omega} \times \vec{r}) + \vec{g} \tag{7.1}$$

where $\vec{a}$ is the specific force acceleration, $\vec{g}$ is the gravitational field strength, $\vec{\omega} \times \vec{v}$ corresponds to the effect of Coriolis force and $\vec{\omega} \times (\vec{\omega} \times \vec{r})$ defines the centripetal acceleration.

To measure these parameters, a typical inertial navigation system consists of three types of sensors: accelerometer, gyroscope and magnetic compass. However, the gyroscope increases the complexity of computations, the power consumption and the total cost. If we assume that the accelerometer is mounted firmly to the vehicle rigid frame and its Y axis points always to the direction of driving, it is possible to infer the velocity only from the accelerometer and compass data. Figure 7.2 illustrates our acceleration integration algorithm. The current velocity vector $\vec{v'}$ is computed by adding the instantaneous acceleration $\vec{a}$ to the vector $\vec{v}$ from the previous step (the size of $\vec{a}$ is intentionally exaggerated in the figure). The magnitude v' becomes:

$$v' = \sqrt{a_x^2 + (v + a_y)^2} \tag{7.2}$$

where a_x and a_y are the components of the acceleration vector. The time step for integration is considered equal to unity.

A known problem of integrating acceleration is the accumulation of errors in time. If no external reference is available, the error can potentially increase to infinity. Our assumption is that the vehicles do not move continuously, but also

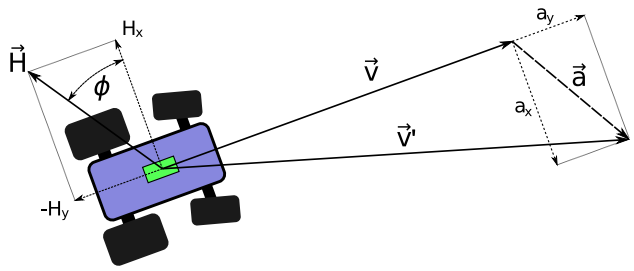


Figure 7.2: Magnetic, velocity and acceleration vectors defined for the vehicle.

have stationary periods during which the velocity estimate can be reset to zero. For determining the "standing still" situation, the sensor nodes continuously analyze the variance of the acceleration over a sliding time window with respect to a threshold determined experimentally. The downside of this approach is that it introduces a certain delay between the actual moment of stopping and the detection of standing still.

7.3.2 Heading Computation

The heading (or azimuth) ϕ is computed from the components of the magnetic field intensity H measured by the magnetic compass (see Figure 7.2):

$$\phi = \arctan(H_y/H_x) \tag{7.3}$$

The heading indicates the vehicle orientation with respect to the magnetic North Pole. [2] In order to map ϕ to the interval $[-\pi : \pi]$, Equation 7.3 becomes:

$$\phi = \begin{cases} \pi + \arctan(H_y/H_x) & \text{if } H_x < 0, H_y \geq 0, \\ \pi/2 & \text{if } H_x = 0, H_y \geq 0, \\ \arctan(H_y/H_x) & \text{if } H_x > 0, \\ -\pi/2 & \text{if } H_x = 0, H_y < 0, \\ -\pi + \arctan(H_y/H_x) & \text{if } H_x < 0, H_y < 0 \end{cases} \tag{7.4}$$

This result holds only when the compass plane is perfectly horizontal. In practice, we must compensate for roll (θ) and pitch (ψ) tilt angles, which can be

[2] The Earth magnetic North is different from the geographic North and, moreover, the angular difference between the two (*declination*) varies with the position of the observation on Earth. This problem does not affect the leader-follower behavior, since all vehicles relate their orientation to the same point – magnetic North.

inferred from the accelerometer output when the system is standing still [164]:

$$\theta = \arctan(\frac{a_y}{\sqrt{a_x^2 + a_z^2}}) \ , \ \psi = \arctan(\frac{a_x}{\sqrt{a_y^2 + a_z^2}}) \tag{7.5}$$

In this case, the transformed magnetic intensities H'_x and H'_y to be used in Equations 7.3 and 7.4 become [56]:

$$\begin{aligned} H'_x &= H_x \cos(\psi) + H_y \sin(\theta)\sin(\psi) - H_z \cos(\theta)\sin(\psi) \\ H'_y &= H_y \cos(\theta) + H_z \sin(\theta) \end{aligned} \tag{7.6}$$

As we can see, the trigonometric computations are quite involved for a sensor node microcontroller. In addition, the variation of roll and pitch angles when the vehicles are moving cannot be measured without a gyroscope.

We take therefore a simpler approach and assume that the tilt will remain constant during driving. As a consequence, the tilt compensation from Equations 7.5 - 7.6 can be replaced by the calibration procedure described in Section 7.7.3. The experimental results from Section 7.8 show that this method is robust to tilt effects whilst driving, as long as the vehicles remain on a relatively flat surface.

7.4 Wireless Communication

The communication protocol required by our approach is straightforward: the leader broadcasts its movement parameters periodically, while the follower just listens for the incoming data packets. This scheme ensures the scalability of the solution by allowing, theoretically, an indefinite number of followers. Nevertheless, a method of logging the behavior of both leader and follower is needed for testing and evaluation purposes. To solve this practical problem, we use the following simple scheme (see Figure 7.3):

- At the end of each sampling period, the leader transmits its movement parameters.
- Upon receiving the message from the leader, the follower sends its own data.

This method has the advantage of avoiding collisions implicitly. The drawback is that scheduling the sampling and communication tasks on the follower becomes problematic (see Section 7.7.5).

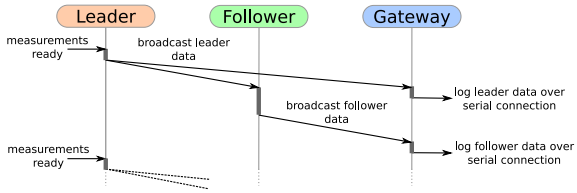


Figure 7.3: Communication protocol sequence.

A gateway node listens to all incoming packets and logs them to a PC. Packet losses are identified based on sequence numbers. In addition to the movement data, the messages sent by the follower include also the controller outputs and raw sensor data. This is a valuable feature because we can reproduce the experiments in the PC-based simulator and correct or tune the controller accordingly.

7.5 Fuzzy Controller

Currently, the field of fuzzy logic is largely overlooked by the WSN community. However, fuzzy logic has several properties that qualifies it as an effective tool for WSN problems. Firstly, it can be implemented on limited hardware and is computationally fast [86, 124]. Secondly, it handles unreliable and imprecise information (as usually the case with sensor data), offering a robust solution to decision fusion under uncertainty [151]. Thirdly, fuzzy-based methodology substantially reduces the design and development time in control systems. Finally, fuzzy controllers handle non-linear systems (most real-life physical systems are non-linear) better when compared to conventional approaches [48].

A fuzzy controller executes three basic steps: *fuzzification*, *inference* and *defuzzification*. During fuzzification, the numeric input values are mapped to fuzzy sets by applying the membership functions. Based on the fuzzified inputs, the controller infers through its IF-THEN rule set and produces an aggregated fuzzy output. The final control action is derived by defuzzifying this aggregated fuzzy output.

7.5.1 Heading and Velocity Control

In our case, the control objective of the follower is to adapt the velocity and heading according to the leader movements. Since the vehicle has separate motors for accelerating and steering, it is reasonable to decompose the problem

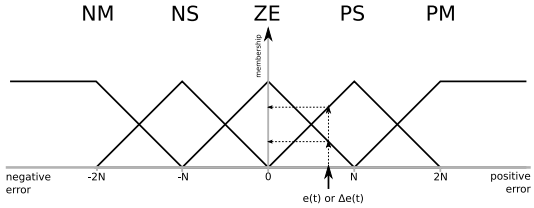


Figure 7.4: Membership functions for fuzzy inputs.

Δe / e	**NM**	**NS**	**ZE**	**PS**	**PM**
NM	PM	PM	PM	PS	ZE
NS	PM	PM	PS	ZE	NS
ZE	PM	PS	ZE	NS	NM
PS	PS	ZE	NS	NM	NM
PM	ZE	NS	NM	NM	NM

Table 7.1: Fuzzy inference rules.

into two independent controllers, with the benefit of simplifying the design and tuning. The two controllers are structurally identical. The only differences lie in the range of the input values, the definition of the membership functions and the post-processing operations.

The two fuzzy controllers follow the design methodology proposed by Mamdani [120]. The building blocks are:

- *Inputs.* As inputs we use the error e and the change in error Δe between the actual values of the movement parameters and the desired values. This is a common approach for improving the controller stability when the output value is close to the optimal operating point.
- *Fuzzy sets.* For an increased control granularity, we define five fuzzy sets for each input, namely NM (Negative Medium), NS (Negative Small), ZE (Zero Equal), PS (Positive Small) and PM (Positive Medium).
- *Membership functions.* To leverage the computational effort, we use triangular membership functions functions spaced equally with distance N, as depicted in Figure 7.4. The width of the membership functions influences the *aggressiveness* of the controller, i.e. what margin of error it tries to achieve.

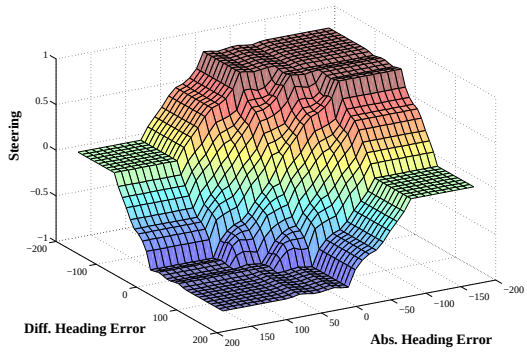


Figure 7.5: 3-D view of the fuzzy controller output.

- *Rule-based inference.* We use the *max-min* fuzzy inference method over the rule set specified in Table 7.1.
- *Output.* The controller output is decided by defuzzifying the aggregated inference result using a simplified centroid method based on discrete output membership functions (for details see also [67]).

As an example, Figure 7.5 shows the decision surface of the simulated heading controller. The steering output corresponds to the vertical axis and ranges from -1 (maximum left turn) to 1 (maximum right turn).

7.5.2 Post-processing

Before actuating the car motors, the outputs of the fuzzy controllers are subjected to a post-processing step, in order to account for the following special driving situations. Firstly, when the vehicle is driving backward, the steering control has to be inverted during the post-processing step. Secondly, when the leader is standing still and the follower velocity gets close to zero, the inductive brake (see also Section 7.7.4) has to be activated. This is a much more efficient braking method than using only the throttle control, as it prevents any forward-backward oscillations.

7.6 Simulation

The simulation framework we developed plays an important role in designing and tuning the fuzzy logic controller. More specifically, we revert to simulation in the following phases:

1. Controller design and evaluation using synthetic, idealized data.
2. Tuning the controller parameters by using real sensor data from experiments as input.
3. Debugging the implementation of the controller on the sensor node platform.

In the following, we describe the first two phases, together with the simulated vehicle model. The third phase is explained in Section 7.7.7. The simulations are cross-validated with field experiments in Section 7.8.4.

7.6.1 Simulation Model

To model the dynamics of the vehicle realistically, we must take both the friction and the throttle capacity of the motor into account. The effective acceleration a_e, i.e. the result of the net force applied to the vehicle, is given by:

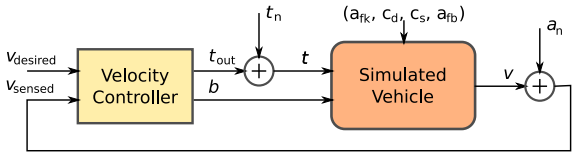
$$a_e = ta_t \ - \ a_f \tag{7.7}$$

where $t \in [-1; 1]$ is the current throttle control value, a_t is the maximum acceleration generated by the throttle (and depends on actual level of the battery powering the motor) and a_f represents the acceleration induced by friction. The effective acceleration a_e is further integrated to estimate velocity, as explained in Section 7.3.1.

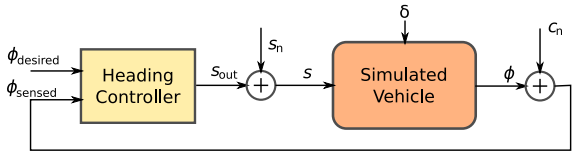
Deriving an accurate model for the frictional acceleration a_f is more complicated due to the multitude of physical factors involved in the process. In our simulations, we use the following simplified model:

$$a_f = a_{fk} \ + \ c_d|v| \ + \ c_s|\delta| \ + \ ba_{fb} \tag{7.8}$$

where a_{fk} is the usual kinetic friction, $c_d|v|$ represents the drag friction (proportional to velocity v for small objects moving at low speeds, according to Stokes law), $c_s|\delta|$ yields the sliding friction when the vehicle is steering with angle δ



(a) Velocity controller



(b) Heading controller

Figure 7.6: Simulated controller models.

and a_{fb} is the friction induced by braking ($b \in \{0; 1\}$ being the brake control signal). [3]

The final component of the vehicle model is the heading. We assume that the front wheels of the vehicle can steer with an arbitrary angle between $-\delta$ and δ. The equation for updating the current heading ϕ' is:

$$\phi' = \phi - s\delta \tag{7.9}$$

where ϕ is the heading at the previous time step and $s \in [-1; 1]$ is the steering control value.

7.6.2 Controller simulation

We first present the separate simulations of the velocity and heading controllers, and then the combined simulation.

Velocity simulation

The velocity simulation follows the model described by Equations 7.7 and 7.8. In Figure 7.6(a) we represent the simulated environment, which accepts the

[3] The heuristic assumption that sliding friction varies proportionally with the steering angle is solely based on our observations from field tests. Coefficients c_d, c_s are also obtained experimentally.

throttle t and brake b control signals from the velocity controller, and yields the simulated velocity v.

We model two types of noise – throttle noise t_n and accelerometer noise a_n – as uniformly distributed random variables. The simulated noisy velocity measurement v_{sensed} is input to the velocity controller. The controller tries to achieve the desired velocity $v_{desired}$ using its throttle t_{out} and brake b outputs.

Heading simulation

Figure 7.6(b) shows the structure of the heading simulation. The heading controller has two inputs – the desired heading $\phi_{desired}$ and the measured heading ϕ_{sensed} – and outputs the steering control value s. The car drives at constant velocity and, with each iteration of the simulation, the current heading ϕ is updated with the current steering control value s using Equation 7.9.

Similar to the velocity simulation, we consider two types of uniformly distributed noise: steering noise s_n (related to inaccuracies in the steering mechanism and external influences, such as a rough road surface) and compass noise c_n. In order to create a more realistic response of the steering, we apply a running average filter at the controller output. This corresponds to the actual behavior of the vehicle steering, which exhibits a certain delay in reacting to high rates of changes.

Figure 7.7 shows the behaviour of the heading controller. The simulation is instructed to change the desired heading 90° to the right every 100 iterations, starting with a heading of 30° . This causes the simulated car to drive an approximately square pattern. At the current instance of time, we see the measured heading, the error e, the change in error Δe and the steering command.

Full simulation

The full simulation includes both the velocity and steering models described in the previous sections. This combination entails that the steering control value of the previous iteration is fed to the friction model of the velocity simulation. The current steering control value is inverted if the velocity becomes negative, to properly simulate backward driving.

Our simulations show that the follower controller works adequately when subjected to the model of dynamics explained in Section 7.6.1. The follower manages to closely keep to the leader trajectory and shows realistic changes in speed when taking turns. For detailed results on how field test results match the simulations, see Section 7.8.4.

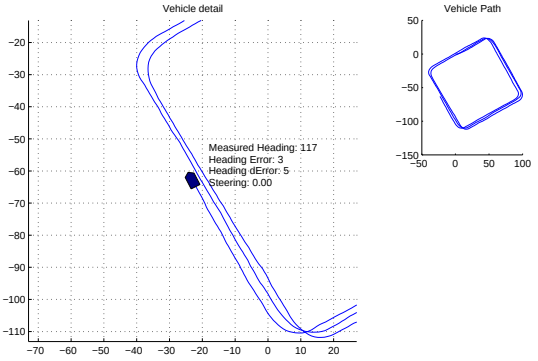


Figure 7.7: Simulation view of the heading controller. The vehicle describes an approximately square pattern on the simulation area (distances represented in meters).

7.6.3 Tuning

For tuning the fuzzy controller membership functions, we run a series of simulations with different controller configurations. For example, Figure 7.8 shows the average absolute error in the heading of a simulated vehicle driving a straight line, plotted against the width of the fuzzy input membership functions. Each point in the plot is the average result of 20 simulation runs. As we explain in Section 7.5, the heading controller becomes more aggressive when the input membership functions of Figure 7.4 get narrower. On the one hand, if the controller is too aggressive, it can produce undesired oscillations in steering. On the other hand, if it is not aggressive enough, the follower has low responsiveness and performs suboptimally. Figure 7.8 provides an indication of where the actual optimum can be found. We currently use a width of 30°for the membership functions presented in Figure 7.4.

7.7 Implementation

We implement the leader-follower system on two toy cars. The follower is modified to allow the sensor node to control its actuators. The leader car remains unmodified apart from the sensor node attachment. Figure 7.9 shows the two vehicles at our test location.

In the following, we provide a description of the hardware and software implementation. We start by presenting the sensors used, the interfacing to

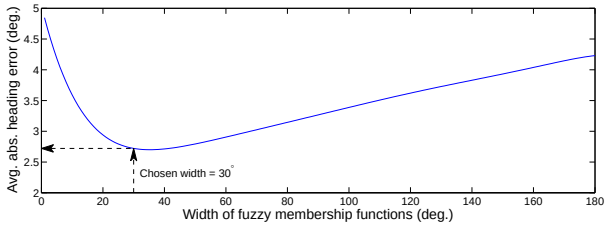


Figure 7.8: Tuning the width of membership functions.

the nodes, the placement on the toy cars and the calibration procedure. Next, we describe the actuation on the follower vehicle, the scheduling on the sensor nodes and the implementation of the calculations from Section 7.3. We end this section with a description of the debugging and benchmarking of the controller implementation.

7.7.1 Sensing Interface

Hardware

As the generic sensor node board, we use the Ambient μNode 2.0 platform [1], running on the low-power MSP430 microcontroller from Texas Instruments (48kB of FLASH memory and 10kB of RAM). The radio transceiver has a maximum data rate of 100kbps. The nodes run AmbientRT [91], a real-time multitasking operating system.

The selection of inertial sensors is driven by cost concerns, accuracy, form factor, power consumption and interfacing capabilities. As accelerometer, we choose the LIS3LV02DQ three-axial sensor from STMicroelectronics [29]. The price is around 15 $ and the typical power consumption is 2mW. The list of features include user selectable full scale of $\pm$2g and $\pm$6g, I^2C/SPI digital interface, programmable threshold for wake-up/free-fall and various sample rates up to 2.56kHz. Finding a low-power, three-axial magnetic compass operating at 3V supply voltage is more difficult. We choose the MicroMag 3 sensor manufactured by PNI Corporation [24]. The price range is 60 $ and the typical power consumption is 1.2mW. The MicroMag3 uses the Magneto-Inductive (MI) sensing technique [57] and provides a relatively large field measurement range ($\pm$11 Gauss) on the digital SPI interface. The compass sensor and the accelerometer are connected to the node using a shared SPI bus.

Figure 7.9: Leader and follower toy cars in the field.

Sensor Drivers

Choosing the right sampling strategy for the two sensors creates an intricate trade-off among accuracy, power consumption and scheduling feasibility. For an accurate velocity integration, it is imperative to sample the acceleration at a rate higher than the Nyquist frequency of any significant noise in the input data. Not following this rule causes aliasing effects that map frequency components of the noise to seemingly arbitrary positions in the spectrum. We establish experimentally the sampling frequency of 160 Hz as an optimal value for our system (see details in the following section). The sampling rate of the compass is also configurable, but special attention is required, because it relates inversely proportional to the achievable sensor resolution and subsequently to the angular sensitivity. We choose to use the sampling frequency of 16 Hz, which provides theoretically an angular sensitivity higher than 0.5° , and additionally matches the control frequency.

Having the accelerometer sampled ten times faster than the compass can generate problems, as both sensors share the same SPI bus of the MSP430 microcontroller. Fortunately, the compass is able to complete its measurements in the background, meanwhile releasing the SPI bus for accelerometer sampling. The only complication is that the compass requires explicit read commands for each of the three axes. As a consequence, the sampling tasks of the two sensors have to be interleaved. The detailed scheduling strategy is explained in Section 7.7.5.

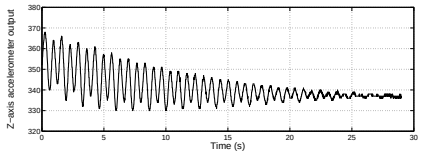
Preliminary Sensor Evaluation

As a first step, we conduct a preliminary evaluation of the sensors, in order to study their accuracy and robustness to external influences. We present in Figure 7.10 several representative tests for each sensor. Figure 7.10(a) shows the accelerometer output when released freely to swing as a pendulum (for clarity only the Z axis output is plotted). The graphic follows closely the actual movement and the noise level is low. In contrast, the second experiment presented Figure 7.10(b) exhibits considerable noise during movement. In this case, the accelerometer is placed on a toy car that accelerates forward and then backward, as suggested by the low-pass filtered signal plotted with dotted line. During stationary periods, though, the sensor output remains stable, with only single bit fluctuations.

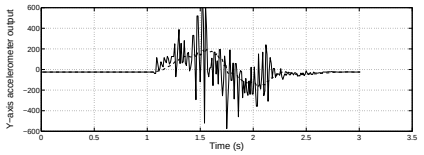
A similar experiment is performed for Figure 7.10(e). The result of the velocity integration from Equation 7.2 is presented at different sampling frequencies supported by the accelerometer. The lower 40 and 80 Hz frequencies exhibit large velocity fluctuations, whereas 160 and 320 Hz provide a velocity progression that correlates well with the experiment. Because the improvement from 160 to 320 Hz is not significant and scheduling becomes more problematic at higher frequencies, we choose the 160 Hz sample frequency for the accelerometer.

Figure 7.10(c) depicts the output of the magnetic compass when rotating around the Z axis. The signal describes a smooth sinusoid recording the intensity of the magnetic field on the Y axis from $-H_y$ to H_y, as expected. In the second test of the compass, the sensor is mounted on a toy car pulled 25 meters through a straight hallway from our building. The hallway is lined with large concrete pillars containing metallic reinforcement. Figure 7.10(d) shows the results of two such experiments, which both confirm strong influences of the nearby metals on the compass sensor. [4] The results clearly show that the compass cannot provide useful measurements when moving close to large metal objects, making indoor use infeasible in most cases. Additional experiments show that the magnetic field caused by the follower's motor has a significant influence on the compass sensor as well. This effect requires the compass to be shielded or placed at sufficient distance from any actuator.

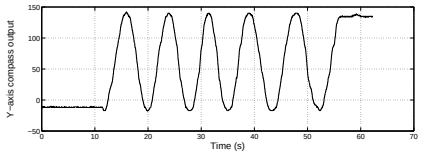
[4] The influence pattern is consistent, as the results of the two experiments correlate well. In Figure 7.10(d), an artificial offset is introduced between the two signals, in order to enhance visibility.



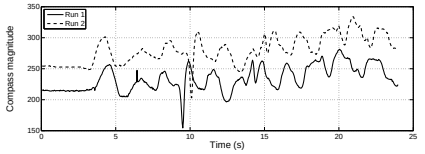
(a) Accelerometer in free pendulum swing.



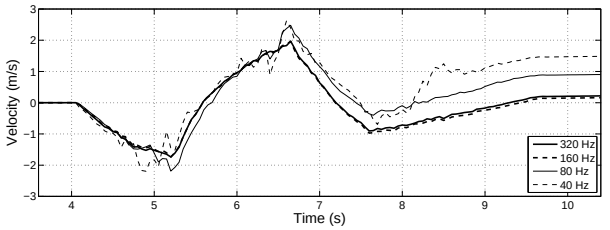
(b) Accelerometer on toy car accelerating forward/backward.



(c) Compass rotating around Z axis.



(d) Compass on toy car moving through hallway.



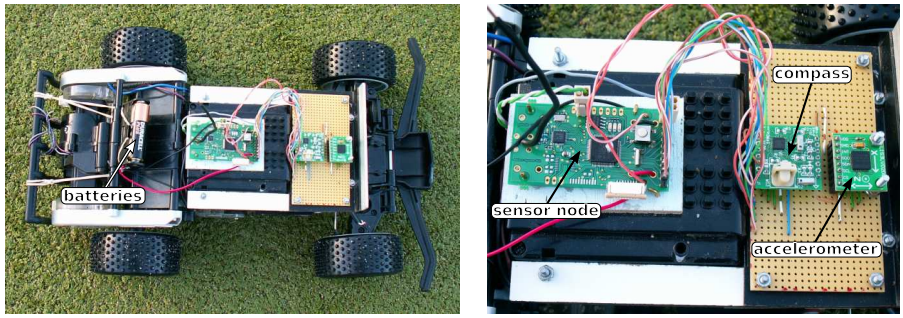
(e) Velocity integration at different frequencies.

Figure 7.10: Sensor evaluation experiments.

7.7.2 Sensor Placement

Sensor placement is an important aspect that can seriously affect the system performance. Both the accelerometer and the compass need to be mounted rigidly to the vehicle frame, in order to minimize the level of vibrations during movement. Because we use different vehicles for leader and follower, the placement of the sensors is not identical. This problem is easily solved through calibration (see Section 7.7.3). As indicated by the preliminary evaluation presented in Section 7.7.1, the placement of the compass sensor requires additional care. During our experiments we have identified the following factors influencing the compass:

- The electromagnetic field created by the car motors. This has a strong



(a) Top view (b) Detail

Figure 7.11: Sensor placement on the follower.

effect on the compass output. To alleviate this phenomenon, we use a 5mm shielding metal plate mounted below the sensor board.

- The batteries of the sensor node. Surprisingly, changing the batteries or even swapping the places of the two batteries has a visible impact on the compass readings. The easiest solution is to place the battery pack as far from the compass as possible.
- The radio transceiver. We could not identify the exact cause of this influence. The most plausible explanation is a combination between the electromagnetic field around the antenna and the significant power drain when the node is transmitting or receiving. Our strategy to avoid this problem is to introduce an additional delay between compass sampling and radio tasks (see scheduling details from Section 7.7.5).

The final sensor placement on the follower car is shown in Figure 7.11.

7.7.3 Calibration

The calibration procedure we utilize compensates for both magnetic field distortions (due to ferrous or magnetic nearby structures) and inclination relative to the reference horizontal plane (see also Section 7.3.2). The idea is to drive the vehicles in a circular movement and collect the readings on the X and Y compass axis. Instead of the ideal circle centered in (0,0), we usually get an offset ellipse. From the minimum and maximum recorded values we determine the

scale and offset calibration coefficients that project the ellipse back to the desired circle (for the details of this method see [56]). The calibrated compass values are subsequently computed as:

$$\begin{aligned} H'_x &= H_x\, X_{scale} + X_{offset} \\ H'_y &= H_y\, Y_{scale} + Y_{offset} \end{aligned} \tag{7.10}$$

7.7.4 Motor Controller

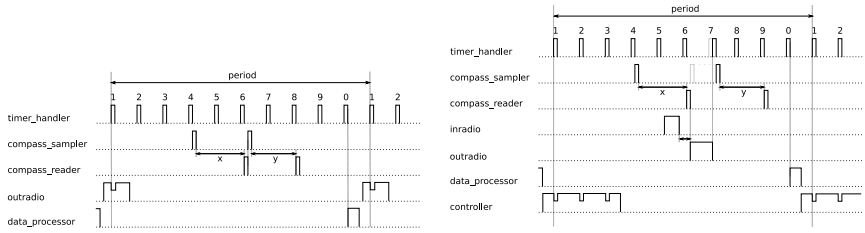
The follower toy car has two electric motors: one for driving and one for steering. To control them, we use a separate MSP430 microcontroller and dedicated circuitry. The two microcontrollers communicate on a separate software I^2C interface, so that the SPI-based dialog with the sensors is not affected. The following control commands are available:

- *Disable*, turns off the motor, the axle can turn freely.
- *Throttle*, controls the intensity of forward or backward acceleration.
- *Steering*, controls the angle of the front wheels.
- *Brake*, short-circuits the rear drive motor to induce inductive drag on the rear wheels.

7.7.5 Scheduling

Getting the right scheduling on the limited sensor node is the most challenging part of the implementation. In this context, the real-time multitasking support of the AmbientRT operating system running on our sensor platform is highly useful. AmbientRT uses earliest deadline first with inheritance (EDFI) scheduling [91], meaning that task priorities are assigned depending on their maximum allowed completion time. As explained in Section 7.7.1, the sampling tasks of the accelerometer and compass must be interleaved. In addition, the influence of radio operation on the compass described in the previous section creates additional dependencies between the tasks associated with these two resources. It is essential therefore to devise the task deadlines in such a way that the execution sequence generated by the EDFI scheduler fulfills all these requirements. [5]

[5] The scheduler verifies the feasibility of the task set, but does not provide further insight into the execution sequence at runtime. The best way to inspect this is to use a digital logic analyzer, which is how we generated Figure 7.12.



(a) Leader (b) Follower

Figure 7.12: Task scheduling.

On the leader, the process is simpler because we know when the radio transmission occurs (at the beginning of a new sampling period) and we do not have a control task (the leader car is driven remotely). Figure 7.12(a) sketches the execution sequence during one sampling period (task durations are not to scale). The heartbeat of the system, termed as the *timer_handler* task, samples the accelerometer at 160 Hz. Being slower, the compass sampling requires two trigger tasks, *compass_sampler* and *compass_reader*, for initiating the measurement and reading the data, respectively. In addition, the X and Y axis cannot be read simultaneously (see also Section 7.7.1). At the end of the sampling period, the *data_processor* task updates the movement status based on the measured data and prepares the message to be transmitted by the *outradio* task.

The scheduling on the follower is complicated by the asynchronous incoming messages from the leader. Moreover, for logging and debugging, we also need an *outradio* task on the follower. As shown in Figure 7.12(b), it is possible to receive data over the radio (the *inradio* task) while sampling the X axis of the compass sensor. In this situation we try to minimize the time overlap between the compass and radio tasks by introducing two delays: (1) the *outradio* task is initiated only after *compass_reader* finishes and (2) the sampling of the Y axis is postponed until the radio transmission is over. The *data_processor* task has a similar function as on the leader, but its completion triggers the *controller* inference and actions described in Section 7.5.

7.7.6 Fixed Point Computation

The calculations presented in Section 7.3 require the implementation of square root and arctangent functions. These are implemented using a simple piece-wise linear representation in a look-up table. The interpolation of values between ta-

ble entries requires fractional number calculations. Since the MSP430 does not provide hardware floating point support, we use fixed point arithmetic exclusively.

7.7.7 Debugging

Debugging the nodes in the field is impractical and therefore we rely on lab reconstruction of failed tests. The follower controller cannot be tested without an environment that responds appropriately to its actions, therefore the input values for the follower controller are obtained from simulation. The leader data recorded from failed field tests is used as input to the simulation to reproduce the situation. In order to find platform-specific problems, we run the software on a gateway sensor node with a serial connection to the PC. The PC provides the input values for the software function under test and reads back the results.

Most of the platform-specific problems relate to different integer size computations. The MSP430 is a 16 bit processor, meaning that the C-compiler generates 16 bit calculations by default. If 32 bit calculations are not specified explicitly, 16 bit calculations are generated for the node and 32 bit calculations are generated for the PC. As a consequence, the software works on the PC and fails on the nodes.

7.7.8 Controller Benchmarking

Using the debugging method presented in Section 7.7.7, we determine how fast the controller and its sub-components execute. Table 7.2 shows the typical execution times. The time spent in the *fuzzification*, *inference* and *defuzzification* stages for the heading and velocity controllers is identical. Notably, the execution of the whole heading controller takes longer than the velocity controller because a modulo function is used to map the heading error inputs to the range $[-\pi, \pi]$ (see Equation 7.4).

7.8 Field Tests and Results

This section outlines the field experiments performed with the system at our university campus. We first test the behavior of the heading controller and subsequently evaluate the complete leader-follower system.

Controller Component	Avg. Execution Time
Full Controller	2.1 ms
Velocity Controller	0.9 ms
Heading Controller	1.2 ms
Fuzzification	0.1 ms
Inference	0.5 ms
Defuzzification	0.2 ms

Table 7.2: Controller benchmark.

7.8.1 Heading Tests

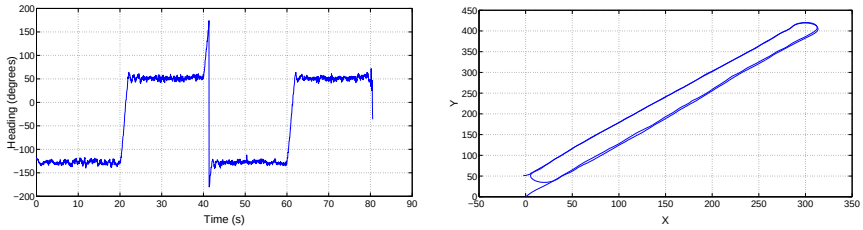
To isolate problems specific to the compass and heading controller, we test the ability of the follower to maintain a specific heading without velocity control or post-processing. The follower starts with a preprogrammed desired heading and changes it with $\pm 180°$ at fixed intervals of time, theoretically following a linear stretch back and forth. This test is performed at our university's running track, oriented approximately 31° NE. The heading change time interval is chosen such that the follower drives over approximately half the length of the track at maximum velocity.

The 180° turns are clearly visible in Figure 7.13(a). Every 20s, the car takes a fast turn to right and thus remains approximately on the same path, as suggested by Figure 7.13(b). In this test, the measured heading deviates 2° on average (with a maximum of 10°) from the desired heading when the car is not actively turning. This result indicates a good behavior of the fuzzy logic heading controller at full speed. However, the performance can deteriorate when the compass inclination changes during driving due to surface irregularities. Statistics extracted from a total of 24 minutes of testing yield an average deviation of 3° , with a maximum of 39° .

The running track tests show that the compass sensor is sensitive to changes of position, orientation and inclination. Changes in placement require re-calibration. Additionally, when the vehicle gets close to large metal objects or faces surface irregularities, it performs worse due to measurement errors. However, when the sensor is correctly calibrated and not otherwise influenced, the follower maintains a constant heading at fixed throttle.

7.8.2 Complete Tests

We experiment with the complete system at our university's hockey field, oriented 12° NE. This location has the advantage of being a large, relatively flat



(a) Heading (b) Integrated position

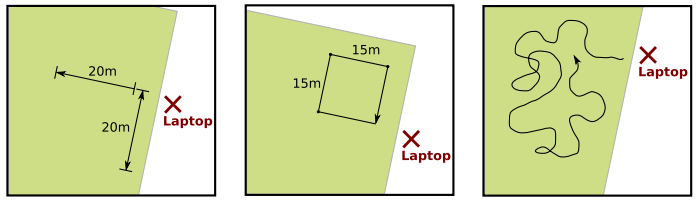
Figure 7.13: Results of a test on the running track.

surface, without metallic structures close by. To evaluate the performance of the system in different situations, we perform tests with three different drive patterns:

- *Linear tests*. The leader drives a 20m straight line.
- *Square tests*. The leader drives a square of side length 15m. The finish line is perpendicular to the start line.
- *Random tests*: The leader drives along a random path for 30s with varying velocity.

The three driving patterns are illustrated in the closeups from Figure 7.14. We perform 20 experiments for each type of test. We vary the driving speed, alternate between driving forward and backward (in random tests), and change the path orientation (in linear tests). In each experiment, the distance between the leader and the follower is 2m at the starting line. At the end of the experiment, we measure the *final heading* of the vehicles (using a regular compass), the *relative distance* between them and the *distance to the finish line* (for linear and square tests). In addition, we log the velocity and heading computed by the two sensor nodes, as well as the throttle and steering outputs of the fuzzy controllers running on the follower. Although not an absolute reference, these logs provide useful information about what is actually happening on the two nodes. Details are presented in the following sections.

The logs also provide the means to calculate the packet loss as identified by missing sequence numbers. The results listed in Table 7.3 show the average percentage of packets lost during the tests. The random tests show less packet loss, possibly due to the fact that these tests were performed closer to the gateway node.



(a) Linear tests (b) Square tests (c) Random tests

Figure 7.14: Schematic view of the hockey field tests.

	Leader	**Follower**
Linear tests	1.3%	1.1%
Square tests	0.6%	1.4%
Random tests	0.2%	0.1%

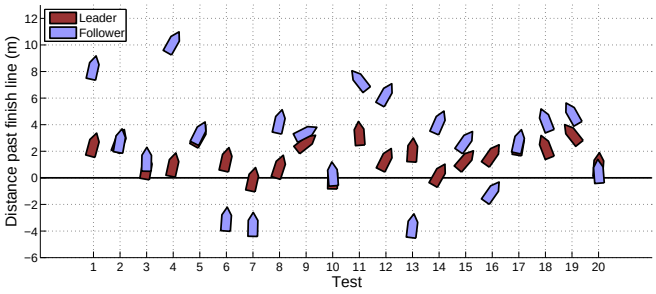
Table 7.3: Packet loss.

Linear Tests

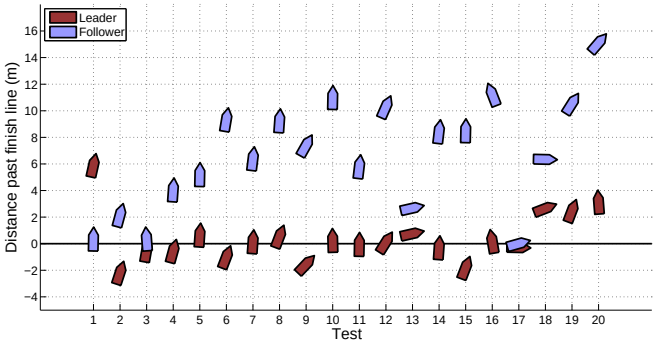
The linear tests are particularly valuable to assess whether the follower can maintain a constant heading while imitating the velocity of the leader. Figure 7.15(a) depicts the ending positions and headings of the two vehicles with respect to the finish line (thick horizontal line at 0m). We see that the follower succeeds to copy the leader movement in most of the experiments. The final relative distance is 3.7m on average, which means that the follower deviates with approximately 9cm per traveled meter (the initial relative distance is subtracted when computing this value). The difference in final headings is 8° on average and in 90% of the cases below 11° .

The results presented in Figure 7.15(a) show only the final situation of each test. Figure 7.16 gives further insight on how the performance of the follower may actually vary in a particular experiment (test 10). The top half shows the velocity as integrated by the two nodes and the corresponding throttle control of the follower, while the bottom half shows the changes in heading and the steering output of the heading controller. We make the following observations:

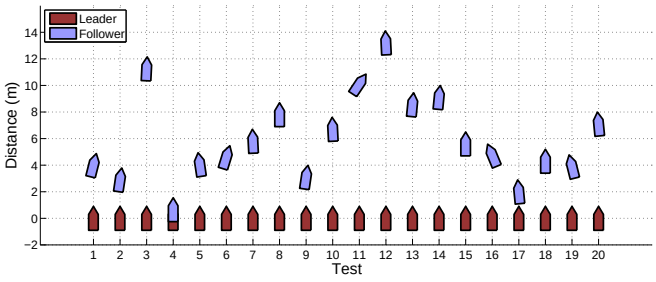
- In the interval 4-8s, the leader accelerates to a velocity higher than what the follower can manage. During the rest of the test, the follower succeeds to keep up. Eventually, the velocity is reset through the motion detection technique explained in Section 7.3.1. The two cars end up very close to



(a) Linear tests.



(b) Square tests.



(c) Random tests.

Figure 7.15: Finish results of the field tests.

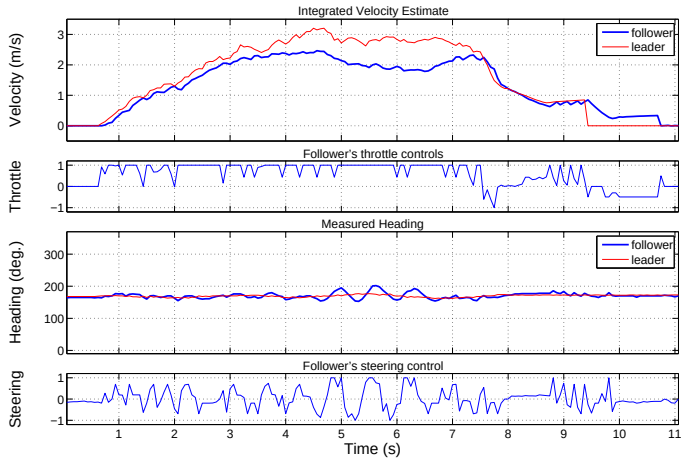


Figure 7.16: Velocity and heading of the 10^{th} linear test.

the finish line: 0.1m and 0.3m.

- Although the final difference in heading is only 6° , the follower sways much more in the interval 4-8s. The resulting excessive steering causes extra friction and reduces the velocity of the follower, which remains behind. This shows that the heading controller can become unstable when driving a straight line, although it corrects the orientation in the end.

Square Tests

Compared to linear driving, during square tests the leader successively steers by 90° and runs over a larger distance (60m in total). Figure 7.15(b) summarizes the results of the 20 experiments. As expected, the differences in position and orientation increase compared to linear tests. The average final distance between vehicles is 8.3m, meaning that the follower deviates approximately 11cm per traveled meter. The difference in final heading is 12° on average and in 90% of the experiments below 20° .

Figure 7.17(a) plots the data transmitted by the sensor nodes during one particular square test (test 6). Gaps in the graphs correspond to occasional packet losses at the gateway. We make the following observations:

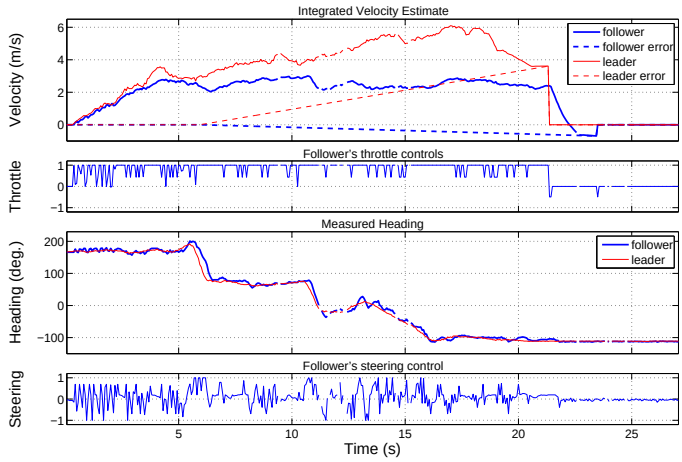
- Both the leader and the follower measure consistently the changes in heading. We can see clear turns corresponding to the rectangle corners at intervals of approximately 5s. A certain amount of swaying by the leader, especially around 3s, cannot be avoided because the leader is driven manually. The final difference in heading between the two is 11° .
- The velocity increases correctly in the first 6s, more precisely until the first turn. From this moment, the leader accumulates errors and ends up with a high value at the moment of stopping. The velocity is reset through the motion detection technique explained in Section 7.3.1. The follower does not exhibit this problem. Its speed remains relatively constant from 6s until 21s, when it finds out that the leader has stopped and, consequently, applies the braking procedure. The follower velocity returns to a value much closer to zero compared to the leader.
- Looking back at Figure 7.15(b), we understand the effect of error accumulation in the leader velocity. The follower tries to keep up and accelerates to its maximum. The leader detects that it stopped with a certain delay and then informs the follower. The latter also needs some time to brake, so it eventually stops at 10.7m away from the leader.

Random Tests

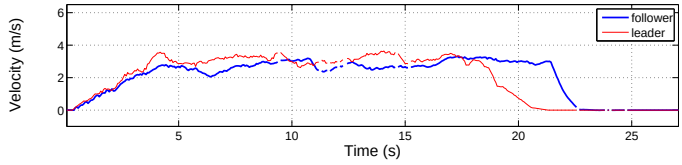
The random tests involve steering in random directions and large variations in velocity. Each of the 20 experiments lasts 30 seconds (there is no finish line). Figure 7.15(c) shows the relative distances and headings of the two vehicles. We obtain an average of 6m final distance between vehicles and a final heading difference of 8° .

Figure 7.18 depicts one particular experiment (random test 17). In this experiment, the leader drives with high variations in velocity and heading. We make the following observations:

- Both the leader and the follower are responsive to the changes in heading. The abrupt transition just after 16s is in fact the trigonometric turning point from -180° to 180° . At this instance, we notice a delay in the response of the follower steering. The final difference in heading between the two vehicles is 6° .
- Both nodes record clearly the steep acceleration and deceleration. The leader velocity accumulates few errors and returns close to zero at the end



(a) Raw data.



(b) Corrected velocity.

Figure 7.17: Velocity and heading of the 6^{th} square test.

of the experiment. The follower has a larger negative offset, which means that he drives faster than he thinks.

- The follower matches the leader velocity during the periods with less steep acceleration, for example around 1s and 16s, but falls behind during high peaks, such as 4s and 12s. The reason lies in the constructive limitations of the follower car, which cannot accelerate as fast as the leader. Overall, the follower moves slower but, since it incurs a delay in braking, it ends up at 2m distance to the leader, which is is approximately the same distance as the cars started at.

Surprisingly, the random tests show better performance than the linear and square tests with respect to the finishing position, although the distance covered is larger. The explanation lies in the steering behaviour of the follower: since

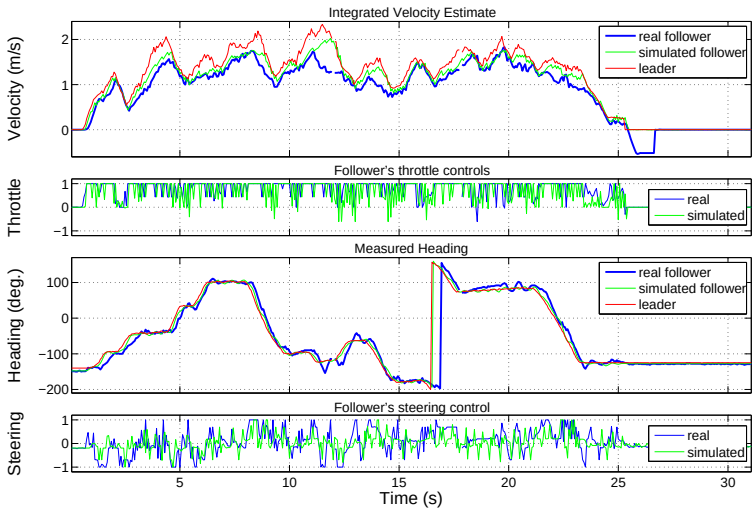


Figure 7.18: Velocity and heading of the 17^{th} random test (including simulation).

the follower does not sway that much, it can attain a higher velocity due to the reduced friction and it can keep up better with the leader. Moreover, since the follower's steering quickly stabilizes to the desired heading, the heading deviation is smaller as well.

7.8.3 Summary of results

Figure 7.19 summarizes the results of final differences in distance and heading. The initial relative distance of 2m between cars is not subtracted from these results. The error bars are plotted with 5 and 95 percentiles.

In order to better characterize the continuous performance of the leader-follower system, Table 7.4 provides detailed statistics from all the logged tests. For both velocity and heading, we compute the mean and standard deviation of the absolute error between the two vehicles at any moment in time, as well as the correlation coefficient [6] of the signals recorded in each test. We make the following observations:

[6]The correlation coefficient is a commonly used measure of similarity between two signals. A correlation coefficient close to 1 indicates well matching signals.

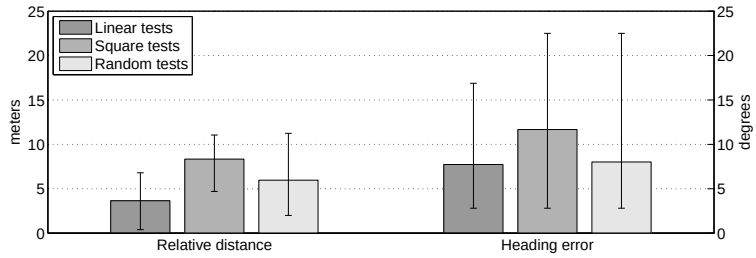


Figure 7.19: Summary of results.

- In contrast to the results of ending positions (from Figure 7.19), the values in Table 7.4 offer just the relative image of what the nodes "think" about their velocity and heading. More specifically, these values include the inherent sensor and integration errors.
- The heading correlation coefficient in linear tests is smaller than in square and random tests (0.64 compared to 0.99). The reason is that the heading varies much less in linear tests, as there are no turns, and therefore any small difference in the compass readings of the two nodes has a strong impact on the correlation.
- The performance in random tests is slightly worse than indicated by the results of ending positions. A detailed analysis of the logs reveals two main reasons. Firstly, during random tests, the leader is subjected to steep acceleration and steering at times. Due to constructive limitations, the follower can not match this behavior, but later on he recovers the difference. Secondly, the vehicles drive longer distances in random tests than in other tests and thus accumulate higher errors through acceleration integration. However, their velocity is physically limited, which explains why the actual ending results are better than what the sensor nodes compute.

7.8.4 Discussion

The results presented so far open several points of discussion. In the following, we briefly examine the effects of error accumulation and error canceling, as well as the accuracy of our simulations with respect to the experimental results.

	Follower vs. Leader			Simulated vs. Follower		
	linear	square	random	linear	square	random
Velocity [m/s]						
mean abs. err.	0.53	1.97	2.75	0.20	0.48	0.58
std. dev.	0.55	1.42	1.93	0.24	0.39	0.34
corr. coeff.	0.93	0.63	0.54	0.96	0.84	0.84
Heading [°]						
mean abs. err.	7.38	10.23	14.99	7.35	9.11	11.38
std. dev.	9.79	14.23	21.06	9.78	11.96	15.45
corr. coeff.	0.64	0.99	0.99	0.64	0.99	0.99

Table 7.4: Test error and correlation statistics.

Error Accumulation

Referring back to Figures 7.15(a) and 7.15(b), we notice that the follower generally ends up further relatively to the finish line than the leader (the random tests are irrelevant in this case, as there is no finish line). As mentioned before, this is caused by the errors accumulating in the acceleration integration process and the inherent delay of the motion detection technique (see Section 7.3.1). The pertinent question that remains is how does the error accumulation look like in practice?

To answer this question, we go back to the square test depicted in Figure 7.17(a). The top plot shows an estimation of the accumulated error in velocity for both leader and follower (dotted lines). The error seems to start accumulating when the cars make the first turn, probably because of the small changes in tilt angles. This is also the moment where the difference in the integrated velocity of the two vehicles starts to be significant. We attribute the worse performance of the leader to the soft suspension and the weaker mounting of the sensor. Figure 7.17(b) shows the corrected velocities, where the linear error estimation is subtracted from the raw data. The matching between the two signals is much better and corresponds to the actual behavior observed in the field.

In conclusion, error accumulation becomes significant due to changes in tilt angles and increases linearly in absolute value. Whether the error can be identified and compensated for at runtime remains an open problem.

Error Canceling

Despite the integration errors at the leader and the swaying behavior of the follower, the latter still manages to keep on the right trajectory, especially when the leader is driven with frequent changes in heading. This effect is shown both during the random tests and in an additional test that lasted for more than 10 minutes. This is partly explained by the fact that errors tend to compensate for one another. In addition, the limited velocity of the vehicles improves the overall performance. When the leader does not drive faster than what the follower can manage, the latter follows correctly, even when the estimate of the leader velocity rises beyond its constructive possibilities.

Simulation validation

The experimental results offer the possibility to validate and refine the simulation vehicle model presented in Section 7.6. For this purpose, we use the data recorded during the field tests as input to the simulation framework. Figure 7.18 shows the simulated velocity and heading of the follower, as compared with the perceived velocity and heading derived from sensor data. We notice that the field results validate the simulation output. The most significant difference occurs in the steering control output, with the real follower steering slower than in simulation. The reason is not the maximum steering angle the vehicle can achieve, but rather the delay between a change in the steering signal from the controller and the actual change in the heading of the vehicle.

Table 7.4 gives a quantitative view on how well the simulation matches the real behavior. Similar to Section 7.8.3, we provide three statistical indicators for each type of test: the mean and standard deviation of the absolute error, and the correlation coefficient of the signals recorded in each test. In general, the real and simulated behaviors match well. The correlation coefficient in linear tests is relatively small for similar reasons as explained in Section 7.8.3. A detailed analysis of individual tests shows that a more accurate friction model is needed to further enhance our simulations.

7.9 Conclusions

We presented a distributed method for team trajectory coordination using regular toy cars augmented with wireless sensor nodes and inertial sensors. As concluding remarks, we briefly iterate the main advantages and limitations of

our method, mention the practical lessons learned and outline the future work directions.

Advantages. The method we described enables autonomous mobile team coordination based solely on compact, low-cost wireless sensors and actuators. The solution is not restricted to vehicles on wheels, but supports *any moving entities capable of determining their velocity and heading*. Therefore, it creates a promising basis for both machine-to-machine and human-to-machine spontaneous interactions in the field. Another advantage is the constructive robustness to errors: from any initial or intermediate faulty orientation, the follower is able to return to the correct heading, due to the usage of a global reference system. Furthermore, making use of a fuzzy controller facilitates the implementation on resource constrained sensor nodes and handles robustly the noisy sensor data as well as the course mechanical capabilities of the vehicles. With respect to wireless communication, the leader-follower model is inherently scalable and tolerates temporary packet losses.

Limitations. As any inertial navigation solution, FollowMe accumulates errors through acceleration integration. Without stopping periods or external reference sources (e.g. GPS), the error can potentially grow to infinity. Additionally, our current implementation does not support dynamic tilt compensation (for example when moving on inclined terrain). Furthermore, the compass utilization is restricted to environments without strong magnetic influences, typically outdoors. With respect to convoy-like applications, the main limitation is the lack of any information and subsequently control of the relative distance between the vehicles (see also future work).

Lessons learned. The main lesson learned during the implementation on sensor nodes is that sampling inertial sensors requires a major share of the available CPU power and preferably a multitasking operating system. Scheduling all the tasks is difficult, especially due to the compass sensitivity to various interferences. The large amount of computation involved in inertial navigation algorithms is another potential source of hard-to-identify bugs. Introducing a feedback loop from the sensor node to the simulator proved to be an efficient debugging method for this problem. From field experiments, the most important lesson learned is that the constructive limitations of the vehicles can have both a negative impact (such as the lack of an actual braking system) and a positive influence (the limited engine power of the follower bounds the effect of error integration in velocity). Compass calibration remains a necessary step for producing accurate experimental results. Driving the toy cars proved to be challenging, as collisions at 30 km/h would destroy the sensors mounted on top. In general, experiments were more time-consuming than expected, required a ded-

icated, large open field and, last but not least, were dependent on the occasional good weather.

Future work. We plan to pursue two directions for future work. One is to incorporate relative distance estimation (for example RSSI information) into the fuzzy controller, in order to apply our solution to convoy-like applications. The other one is to explore group interactions among heterogeneous moving entities (not only vehicles on wheels) using the general FollowMe idea.

Chapter 8

Conclusions

At the time of writing this thesis, a search for "wireless sensor networks" in Google Scholar retrieves more than 12,000 articles published in the last year. This corresponds roughly to almost one thousand new publications per month, as indexed by Google. Looking back from year 2000, Figure 8.1 shows an exponentially increasing interest in the field. The research community is more active than ever in pushing WSNs and related technologies in the future "ubiquitous" world.

Even more interesting is the evolution of WSNs in the industrial arena. The reserved enthusiasm from several years ago transformed gradually into concrete initiatives at all levels. Major standardization efforts are being put into IEEE 802.15.4 and ZigBee. Leading semiconductor companies, such as Texas Instruments and Freescale, release at accelerated pace new integrated solutions for ultra low-power networking. Developing miniaturized, low-cost and low-power sensors to be interfaced with such wireless platforms receives increasing attention within high profile manufacturers, such as Honeywell and STMicroelectronics. Key players from industrial automation – Siemens, Schneider Electric – explore the wireless PLC alternative. At the end of the chain, business solutions providers, such as SAP and IBM, unlock new functionalities by combining WSNs and RFIDs in supply chain management and asset tracking. Even car manufacturers, as BMW, study new opportunities of enhancing Car2X communication and services.

These few examples already give a feeling of the extreme market dynamics. The general trend strongly suggests that the epoch of pioneering research in building and experimenting with "motes" is approaching an end. It is now the

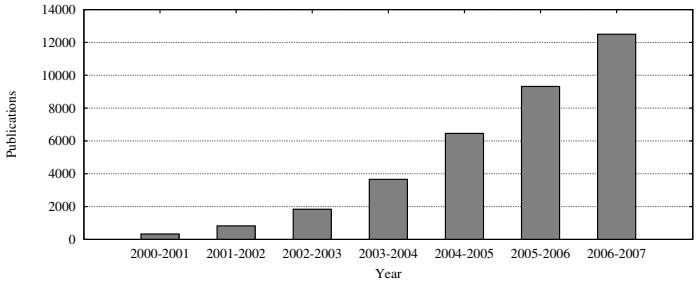


Figure 8.1: Published articles on WSNs as retrieved by Google Scholar.

logical time for *system integration* and for *creating bridges to connected fields*. These are the directions in which we like to think this thesis advances beyond state of the art. To recapitulate, the specific contributions of this thesis are:

- **Market analysis** of several fields that are likely to adopt WSN solutions on a large scale: enterprise systems, transport and logistics, industrial automation, safety-critical processes, automotive industry and automatic meter reading. In each case, we present the market status and predicted evolution, the perspectives of WSNs and the most important requirements and unsolved practical challenges. This contribution corresponds to research question 2 from Chapter 1.
- **Service-oriented architecture** for integrating different WSN platforms and exposing their functionality in a uniform way to the back-end system. To our knowledge, this is the first service-oriented architecture that takes multi-platform WSNs up to the enterprise level. In the European project CoBIs, we demonstrated the complete system, including two WSN platforms, business rules support, reconfiguration via reliable multicast data dissemination, uniform gateway translation using UPnP and final integration into SAP enterprise software. This contribution answers, therefore, research question 3.
- **Reliable data and code dissemination** – a multicast protocol for disseminating data reliably to groups of nodes with minimal energy expenditure. The novelty of this work consists in addressing both end-to-end and local error control, and in applying the multicast communication model for increased scalability and flexibility. Additionally, we give an in-depth analysis of performance factors and tradeoffs considering the entire net-

work stack, while related work usually evaluates transport protocols decoupled from the underlying layers.

- **Rule-based inference** for distributed situation assessment and event detection at the point of action. This chapter brings two innovations to the field: the first explicit use of business rule support and the first distributed fuzzy logic inference engine for wireless sensor nodes. Together with reliable data and code dissemination, this contribution addresses research question 4.
- **Distributed activity recognition** using fuzzy-enabled WSNs that become aware of the user actions and provide context-aware assistance. This work builds on the previous experiments of Amft et al. [39]. The main novelties are to apply fuzzy inference to the unreliable sensor data and to incorporate temporal order knowledge about the sequences of operations, thus increasing the overall accuracy of the recognition system.
- **Mobile team coordination** – a low-cost, low-power method for movement coordination based on inertial sensing, wireless communication and fuzzy control. To our knowledge, this is the first solution considering solely inertial sensors and running entirely on resource-constrained sensor nodes. We provide a complete report on all development phases, covering design, simulation, controller tuning, inertial sensor evaluation, calibration, scheduling, fixed-point computation, debugging, benchmarking, field experiments and lessons learned. Together with distributed activity recognition, this contribution addresses research question 5.

As future work, the most exciting research topic is, in our opinion, the combination of WSNs, artificial intelligence and robotics toward what would represent the ultimate cognitive systems. Many necessary ingredients are already there: wireless communication, miniaturized, low-power sensors, control systems, machine vision, evolutionary algorithms, just to mention a few. Others are still lacking or just in initial development stages: sensor-actuator coordination, distributed training and learning, energy harvesting, multi-modal interfaces, etc. If successful, such a versatile system will enable the two visionary concepts that Chapters 6 and 7 have already sketched: augmented human-machine interaction and intelligent machine-machine coordination.

Bibliography

[1] Ambient Systems. http://www.ambient-systems.net.

[2] AWARE project. http://grvc.us.es/aware.

[3] Clean water act. http://www.waterboards.ca.gov.

[4] Collaborative Business Items (CoBIs). http://www.cobis-online.de.

[5] Controller Area Network. http://www.can.bosch.com.

[6] Coronis Systems. http://www.coronis.com.

[7] Crossbow Technology. http://www.xbow.com.

[8] Energy-Efficient Sensor Networks (EYES). http://eyes.eu.org.

[9] Energy policy act of 2005. http://www.doi.gov.

[10] Equipment and Protective systems intended for use in Potentially Explosive Atmospheres (ATEX) Directive 94/9/EC. http://ec.europa.eu/enterprise/atex.

[11] ExScal: Extreme scale wireless sensor networking project. http://cast.cse.ohio-state.edu/exscal/.

[12] Fieldbus Foundation. http://www.fieldbus.org.

[13] Fire Information and Rescue Equipment (FIRE). http://fire.me.berkeley.edu.

[14] FlexRay Communications System. http://www.flexray.com.

[15] IEEE standard 802.15.4. http://standards.ieee.org.

[16] ITU Internet Reports 2005: The Internet of Things 2005, 7th edition. http://www.itu.int/publ/S-POL-IR.IT-2005/.

[17] Media Oriented Systems Transport (MOST) standard. http://www.mostcooperation.com.

[18] Moteiv Tmote platform, migrated to Sentilla. http://www.sentilla.com.

[19] National Semiconductor LM92 temperature sensor. http://www.national.com.

[20] nRF905 single-chip radio transceiver for the 433/868/915 MHz ISM band - Nordic Semiconductor. http://www.nordicsemi.com.

[21] OMNeT++. http://www.omnetpp.org.

[22] Organisation Internationale des Constructeurs dAutomobiles (OICA). http://www.oica.net.

[23] Particle Computer. http://www.particle-computer.de.

[24] PNI Corporation. http://www.pnicorp.com.

[25] PROFIBUS. http://www.profibus.com.

[26] Reconfigurable Ubiquitous Networked Embedded Systems (RUNES). http://www.ist-runes.org.

[27] Sentilla Corporation. http://www.sentilla.com.

[28] Smart Surroundings. http://www.smart-surroundings.org.

[29] STMicroelectronics. http://www.st.com.

[30] WearIT@work Project. http://www.wearitatwork.com.

[31] ZigBee Alliance. http://www.zigbee.org.

[32] International energy outlook. Technical Report DOE/EIA-0484, Energy Information Administration, 2007.

[33] Active rfid and sensor networks 2008-2018. Technical report, Research and Markets, 2008.

[34] I. Akyildiz and I. Kasimoglu. Wireless sensor and actor networks: Research challenges. *Ad Hoc Networks Journal*, 2(4):351–367, 2004.

[35] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[36] A. Albert. Comparison of event-triggered and time-triggered concepts with regard to distributed control systems. In *Embedded World*, pages 235–252, 2004.

[37] J. Allred, A.B. Hasan, S. Panichsakul, W. Pisano, P. Gray, R. Han J. Huang, D. Lawrence, and K. Mohseni. Sensorflock: an airborne wireless sensor network of micro-air vehicles. In *SenSys '07*, pages 117–129. ACM, 2007.

[38] O. Amft, H. Junker, and G. Tröster. Detection of eating and drinking arm gestures using inertial body-worn sensors. In *International Symposium on Wearable Computers (ISWC)*, pages 160–163, 2005.

[39] O. Amft, C. Lombriser, T. Stiefmeier, and G. Tröster. Recognition of user activity sequences using distributed event detection. In *European Conference on Smart Sensing and Context (EuroSSC)*, pages 126–141, 2007.

[40] ARC Advisory Group. Industrial ethernet infrastructure worldwide outlook. http://www.arcweb.com.

[41] ARC Advisory Group. Safety & critical control system worldwide outlook. http://www.arcweb.com.

[42] ARC Advisory Group. Total automation for process industries worldwide. http://www.arcweb.com.

[43] M. Beigl, A. Krohn, T. Zimmer, C. Decker, and P. Robinson. Awarecon: Situation aware context communication. In *UbiComp*, pages 132–139, 2003.

[44] Michael Beigl, Hans-W. Gellersen, and Albrecht Schmidt. Mediacups: experience with design and use of computer-augmented everyday artefacts. *Computer Networks*, 35(4):401–409, 2001.

[45] G. Beni. From Swarm Intelligence to Swarm Robotics. *Swarm Robotics: SAB 2004 International Workshop*, 2005.

[46] Dimitri Bertsekas and Robert Gallager. *Data Networks (2nd Ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992.

[47] J. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 1981.

[48] J. Bih. Paradigm shift - an introduction to fuzzy logic. *IEEE Potentials*, 25(1):6–21, 2006.

[49] O. Bondarenko, S. Kininmonth, and M. Kingsford. Coral reef sensor network deployment for collecting real time 3-d temperature data with correlation to plankton assemblages. In *International Conference on Sensor Technologies and Applications (SensorComm)*, pages 204–209, 2007.

[50] C. Bornhövd, T. Lin, S. Haller, and J. Schaper. Integrating smart items with business processes: An experience report. In *International Conference on System Sciences (HICCS)*. IEEE Computer Society, 2005.

[51] Athanassios Boulis, Chih-Chieh Han, Roy Shea, and Mani B. Srivastava. Sensorware: Programming sensor networks beyond code update and querying. *Pervasive and Mobile Computing*, 3(4):386–412, 2007.

[52] M. Brand. The "inverse hollywood problem": From video to scripts and storyboards via causal analysis. In *AAAI/IAAI*, pages 132–137, 1997.

[53] M. Buckland and F. Gey. The relationship between recall and precision. *Journal of the American Society for Information Science*, 45(1):12–19, 1999.

[54] R. W. Bukowski, J. D. Averill, R. D. Peacock, and P. A. Reneke. Performance of home smoke alarms. Technical Report 1455, NIST, 2004.

[55] W. Burgard, M. Moors, C. Stachniss, and F.E. Schneider. Coordinated multi-robot exploration. *IEEE Transactions on Robotics*, 21(3):376–386, 2005.

[56] M. Caruso. Applications of magnetic sensors for low cost compass systems. *IEEE Position Location and Navigation Symposium*, pages 177–184, 2000.

[57] M. Caruso, T. Bratland, C. Smith, and R. Schneider. A New Perspective on Magnetic Field Sensing. *Sensors*, 15(12):34–46, 1998.

[58] A. Cerpa, J. Wong, L. Kuang, M. Potkonjak, and D. Estrin. Statistical model of lossy links in wireless sensor networks. In *IPSN*, page 11, 2005.

[59] L. Chaves, L. Sá de Souza, J. Müller, and J. Anke. Service lifecycle management infrastructure for smart items. In *Proceedings of the international workshop on Middleware for sensor networks (MidSens)*, pages 25–30, 2006.

[60] C. Chi and M. Hatler. Industrial wireless sensor networking. Technical report, ON World, 2004.

[61] C. Chi and M. Hatler. Wireless sensor networking for amr & submetering. Technical report, ON World, 2004.

[62] H. Chiang, L. Ma, J. Perng, B. Wu, and T. Lee. Longitudinal and lateral fuzzy control systems design for intelligent vehicles. *Networking, Sensing and Control, 2006. ICNSC '06. Proc. of the 2006 IEEE Int. Conf. on*, pages 544–549, 23-25 April 2006.

[63] C.-Y. Chong and S. P. Kumar. Sensor networks: evolution, opportunities, and challenges. *Proceedings of the IEEE*, 91(8):1247–1256, 2003.

[64] T. Cleary and K. Notarianni. Distributed sensor fire detection. In *International Conference on Automatic Fire Detection*, 2001.

[65] T. Cleary and T. Ono. Enhanced residential fire detection by combining smoke and co sensors. In *International Conference on Automatic Fire Detection*, 2001.

[66] A. Costa, A. De Gloria, F. Giudici, and M. Olivieri. Fuzzy logic microcontroller. *IEEE Micro*, 17(1):66–74, 1997.

[67] A. Dannenberg. Fuzzy logic motor control with msp430x14x. Technical Report SLAA235, Texas Instruments, 2005.

[68] T. H. Davenport. Putting the enterprise into the enterprise system. *Harvard Business Review*, 76(4):121–131, 1998.

[69] L. Sá de Souza, P. Spiess, D. Guinard, M. Köhler, S. Karnouskos, and D. Savio. SOCRADES: A web service based shop floor integration infrastructure. In *The Internet of Things*, pages 50–67, 2008.

[70] Budhaditya Deb, Sudeept Bhatnagar, and Badri Nath. Reinform: Reliable information forwarding using multiple paths in sensor networks. In *LCN '03: Proceedings of the 28th Annual IEEE International Conference on Local Computer Networks*, page 406, 2003.

[71] Adam Dunkels. Full tcp/ip for 8-bit architectures. In *MobiSys '03: Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 85–98, 2003.

[72] China economic review. China third party logistics market report, 2006. http://www.chinaeconomicreview.com/logistics/2006/.

[73] E. Egea-Lopez, A. Martinez-Sala, J. Vales-Alonso, J. Garcia-Haro, and J. Malgosa-Sanahuja. Wireless communications deployment in industry: a review of issues, options and technologies. *Computers in Industry*, 56(1):29–53, 2005.

[74] S. Eisenman, N. Lane, E. Miluzzo, R. Peterson, G. Ahn, and A. Campbell. Metrosense project: People-centric sensing at scale. In *ACM SenSys Workshop on World-Sensor-Web (WSW)*, 2006.

[75] J. Espinosa, J. Vandewalle, and V. Wertz. *Fuzzy logic, identification and predictive control*. Springer-Verlag, 2004.

[76] 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)*, pages 263–270, 1999.

[77] EVD. China: logistiek. http://www.evd.nl.

[78] Sally Floyd, Van Jacobson, Ching-Gung Liu, Steven McCanne, and Lixia Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, 5:784–803, 1997.

[79] H. Fritz. Longitudinal and lateral control of heavy duty trucks for automated vehicle following in mixed traffic: experimental results from the chauffeur project. *Control Applications, 1999, Proceedings*, 2:1348–1352, 1999.

[80] E. Gaura and R. Newman. Wireless sensor networks: The quest for planetary field sensing. In *SenseApp*, 2006.

[81] F. Gemperle, C. Kasabach, J. Stivoric, M. Bauer, and R. Martin. Design for wearability. In *International Symposium on Wearable Computers (ISWC)*, pages 116–123, 1998.

[82] D. T. Gottuk, M. J. Peatross, R. J. Roby, and C. L. Beyler. Advanced fire detection using multi-signature alarm algorithms. *Fire Safety Journal*, 37:381–394, 2001.

[83] Y. Gsottberger, X. Shi, G. Stromberg, T. Sturm, and W. Weber. Embedding low-cost wireless sensors into universal plug and play environments. In *1st European Workshop on Wireless Sensor Networks (EWSN)*, pages 291–306, 2004.

[84] D. Hayat. Traffic regulation system for the future automated highway. *Systems, Man and Cybernetics*, 3, 2002.

[85] D. Hayat. Traffic regulation system for the future automated highway. *Systems, Man and Cybernetics*, 3, 2002.

[86] S. J. Henkind and M.C. Harrison. An analysis of four uncertainty calculi. *IEEE Transactions on Systems, Man and Cybernetics*, 18(5):700–714, 1988.

[87] N. Hodge, L. Shi, and M. Trabia. A distributed fuzzy logic controller for an autonomous vehicle. *J. Robot. Syst.*, 21(10):499–516, 2004.

[88] L. Van Hoesel. *Schedule-based medium access control protocols for wireless sensor networks*. PhD thesis, University of Twente, 2007.

[89] L. Van Hoesel, T. Nieberg, J. Wu, and P. Havinga. Prolonging the lifetime of wireless sensor networks by cross-layer interaction. *IEEE Wireless Communication Magazine*, 12 2004.

[90] Robert D. Hof. The quest for the next big thing. *Business Week*, pages 91–94, 2003.

[91] T. Hofmeijer, S. Dulman, P. G. Jansen, and P. J. M. Havinga. AmbientRT - real time system software support for data centric sensor networks. In *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pages 61–66. IEEE Computer Society Press, 2004.

[92] M. Horsman, M. Marin-Perianu, P. Jansen, and P. Havinga. A simulation framework for evaluating complete reprogramming solutions in wireless sensor networks. In *International Symposium on Wireless Pervasive Computing (ISWPC)*, 2008.

[93] L. Huang. *Reliable Bulk Data Dissemination in Sensor Networks*. PhD thesis, George Mason University, 2007.

[94] Jonathan W. Hui and David Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *SenSys '04*, pages 81–94, NY, USA, 2004. ACM Press.

[95] Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, John Heidemann, and Fabio Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Trans. Netw.*, 11(1):2–16, 2003.

[96] Y. Ivanov and A. Bobick. Recognition of visual activities and interactions by stochastic parsing. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):852–872, 2000.

[97] S. Jacobson, J. Shepherd, M. DAquila, and K. Carter. The ERP market sizing report, 2006-2011. Technical Report AMR-R-20495, AMR Research, 2007.

[98] J. S. Roger Jang. Anfis: Adaptive-network-based fuzzy inference system. *IEEE Transactions on Systems, Man, and Cybernetics*, 23:665–684, 1993.

[99] J. Kacprzyk. Group decision making with a fuzzy linguistic majority. *Fuzzy Sets and Systems*, 18(2):105–118, 1986.

[100] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: mobile networking for smart dust. In *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 271–278, 1999.

[101] S. Kaul, K. Ramachandran, P. Shankar, S. Oh, M. Gruteser, I. Seskar, and T. Nadeem. Effect of antenna placement and diversity on vehicular network communications. In *IEEE Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, pages 112–121, 2007.

[102] R.M. Kling. Intel motes: advanced sensor network platforms and applications. In *IEEE MTT-S International Microwave Symposium*, 2005.

[103] K. Kodagoda, W. Wijesoma, and E. Teoh. Fuzzy speed and steering control of an agv. *Control Systems Technology, IEEE Transactions on*, 10(1):112–120, Jan 2002.

[104] A. Konar and L. Jain. *Cognitive Engineering - A Distributed Approach to Machine Intelligence*. Springer-Verlag, 2007.

[105] N. Kottege and U.R. Zimmer. Relative localization for AUV swarms. In *International Symposium on Underwater Technology*, 2007.

[106] S. Kulkarni and M. Arumugam. Infuse: A TDMA based data dissemination protocol for sensor networks. *International Journal of Distributed Sensor Networks*, 2:55–78, 2006.

[107] Sandeep S. Kulkarni and Limin Wang. MNP: Multihop network reprogramming service for sensor networks. Technical Report MSU-CSE-04-19, Department of Computer Science, Michigan State University, 2004.

[108] D. Lal, A. Manjeshwar, F. Herrmann, E. Uysal-Biyikoglu, and A. Keshavarzian. Measurement and characterization of link quality metrics in energy constrained wireless sensor networks. In *Global Telecommunications Conference*, pages 446–452, 2003.

[109] K. Langendoen and G. Halkes. *Embedded Systems Handbook*, chapter Energy-Efficient Medium Access Control. CRC press, 2005.

[110] G. Leen and D. Heffernan. Vehicles without wires. *Computing and Control Engineering Journal*, 12(5):205–211, 2001.

[111] J. Leohold. Communication requirements for automotive systems. In *Keynote at the 5th Workshop on Factory Communication Systems (WFCS)*, 2004.

[112] Brian Neil Levine and J.J. Garcia-Luna-Aceves. A comparison of reliable multicast protocols. *Multimedia Syst.*, 6:334–348, 1998.

[113] P. Levis and D. Culler. Maté: a tiny virtual machine for sensor networks. In *International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 85–95, 2002.

[114] P. Levis, D. Gay, and D. Culler. Bridging the gap: Programming sensor networks with application specific virtual machines. Technical Report UCB//CSD-04-1343, UC Berkeley, 2004.

[115] Philip Levis, Neil Patel, David Culler, and Scott Shenker. Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, page 2, 2004.

[116] S. Li, Y. Lin, S. H. Son, J. A. Stankovic, and Y. Wei. Event detection services using data service middleware in distributed sensor networks. *Telecommun Syst*, 26(2):351–368, December 2004.

[117] C. Liang, R. Musaloiu-Elefteri, and A. Terzis. Typhoon: A reliable data dissemination protocol for wireless sensor networks. In *European Workshop on Wireless Sensor Networks (EWSN)*, pages 268–285, 2008.

[118] Ting Liu, Christopher M. Sadler, Pei Zhang, and Margaret Martonosi. Implementing software on resource-constrained mobile sensors: Experiences with impala and ZebraNet. In *MobiSYS '04*, pages 256–269. ACM Press, 2004.

[119] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TinyDB: An acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 30:122–173, 2005.

[120] E. H. Mamdani and S. Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies*, 7(1):1–13, 1975.

[121] S. Mann. Wearable computing: a first step toward personal imaging. *Computer*, 30(2):25–32, 1997.

[122] M. Marin-Perianu and P. Havinga. Experiments with reliable data delivery in wireless sensor networks. In *Intelligent Sensors, Sensor Networks and Information Processing Conference (ISSNIP)*, pages 109–114, 2005.

[123] M. Marin-Perianu and P. Havinga. RMD: Reliable multicast data dissemination within groups of collaborating objects. In *Local Computer Networks (LCN)*, pages 656–663, 2006.

[124] M. Marin-Perianu and P. Havinga. D-FLER: A distributed fuzzy logic engine for rule-based wireless sensor networks. In *International Symposium on Ubiquitous Computing Systems (UCS)*, pages 86–101, 2007.

[125] M. Marin-Perianu, T. Hofmeijer, and P. Havinga. Implementing business rules on sensor nodes. In *11th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 292–299, 2006.

[126] M. Marin-Perianu, C. Lombriser, O. Amft, P. Havinga, and G. Tröster. Distributed activity recognition with fuzzy-enabled wireless sensor networks. In *International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2008.

[127] M. Marin-Perianu, N. Meratnia, P. Havinga, L. Moreira Sá de Souza, J. Müller, P. Spieß, S. Haller, T. Riedel, C. Decker, and G. Stromberg. Decentralized enterprise systems: A multi-platform wireless sensor networks approach. *IEEE Wireless Communications*, 14(6):57–66, 2007.

[128] M. Marin-Perianu, N. Meratnia, M. Lijding, and P. Havinga. Being aware in wireless sensor networks. In *15th IST Mobile & Wireless Communication Summit, Capturing Context and Context Aware Systems and Platforms Workshop*, 2006.

[129] R. S. Marin-Perianu, M. Marin-Perianu, P. Havinga, and J. Scholten. Movement-based group awareness with wireless sensor networks. In *5th International Conference on Pervasive Computing (Pervasive)*, pages 298–315, 2007.

[130] J. M. Mendel. Fuzzy logic systems for engineering: a tutorial. *Proceedings of the IEEE*, 83:345–377, 1995.

[131] F. Mondada, L. Gambardella, D. Floreano, S. Nolfi, J. Deneuborg, and M. Dorigo. The cooperation of swarm-bots: physical interactions in collective robotics. *Robotics & Automation Magazine, IEEE*, 12(2):21–28, June 2005.

[132] C. Siva Ram Murthy and B.S. Manoj. *Ad Hoc Wireless Networks: Architectures and Protocols*. Pearson Education, 2004.

[133] V. Naik, A. Arora, P. Sinha, and H. Zhang. Sprinkler: A reliable and energy efficient data dissemination service for extreme scale wireless networks of embedded devices. *IEEE Transactions on Mobile Computing*, 6(7):777–789, 2007.

[134] P. Neumann. Communication in industrial automation - what is going on? *Control Engineering Practice*, 15(11):1332–1347, 2007.

[135] T. Nolte, H. Hansson, and L. Lo Bello. Automotive communications – past, current and future. In *10th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, page 8, 2005.

[136] K. Obraczka. Multicast transport protocols: A survey and taxonomy. *IEEE Communications Magazine*, 36(1):94–102, 1998.

[137] L. O'Brien. Demand-side conditions bode well for 2007 automation market growth. *Hydrocarbon Processing*, November 2006.

[138] V. Osmani, S. Balasubramaniam, and D. Botvich. Self-organising object networks using context zones for distributed activity recognition. In *International Conference on Body Area Networks (BodyNets)*, 2007.

[139] Seung-Jong Park, Ramanuja Vedantham, Raghupathy Sivakumar, and Ian F. Akyildiz. A scalable approach for reliable downstream data delivery in wireless sensor networks. In *MobiHoc '04: Proceedings of the 5th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 78–89. ACM Press, 2004.

[140] Sridhar Pingali, Don Towsley, and James F. Kurose. A comparison of sender-initiated and receiver-initiated reliable multicast protocols. In *Proceedings of the Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 221–230, New York, NY, USA, 1994. ACM Press.

[141] R. D. Poor. *Handbook of RF and Wireless Technologies*, chapter Reliable Wireless Networks for Industrial Applications, pages 401–416. Elsevier, 2004.

[142] J. B. Predd, S. R. Kulkarni, and H. V. Poor. Distributed learning in wireless sensor networks. *IEEE Signal Processing Magazine*, 23(4):56–69, July 2006.

[143] J. Pugh and A. Martinoli. Small-Scale Robot Formation Movement Using a Simple On-Board Relative Positioning System. In *International Symposium on Experimental Robotics 2006*, 2006.

[144] Ruthann Quindlen. *Confessions of a Venture Capitalist: Inside the High-Stakes World of Start-up Financing.* Business Plus, 2001.

[145] D. W. Rebitzer. *Europe Real Estate Yearbook 2007*, chapter The European Logistics Market. Europe Real Estate, 2006.

[146] Niels Reijers and Koen Langendoen. Efficient code distribution in wireless sensor networks. In *WSNA '03: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, pages 60–67, 2003.

[147] E. M. Rogers. *Diffusion of Innovations (4th edition).* Free Press, 1995.

[148] T. J. Ross. *Fuzzy Logic with Engineering Applications.* Wiley, 2004.

[149] T. A. Runkler and M. Glesner. Decade - fast centroid approximation defuzzification for real time fuzzy control applications. In *ACM Symposium on Applied Computing (SAC '94)*, pages 161–165, 1994.

[150] V. Saligrama, M. Alanyali, and O. Savas. Distributed detection in sensor networks with packet losses and finite capacity links. *IEEE T Signal Proces*, 54(11):4118–4132, November 2006.

[151] V. N. S. Samarasooriya and P. K. Varshney. A fuzzy modeling approach to decision fusion under uncertainty. *Fuzzy Sets and Systems*, 114(1):59–69, 2000.

[152] Yogesh Sankarasubramaniam, Özgür B. Akan, and Ian F. Akyildiz. ESRT: Event to sink reliable transport in wireless sensor networks. In *MobiHoc '03*, pages 177–188. ACM Press, 2003.

[153] H. Schaap. Wireless sensor network standard for logistic processes. Master's thesis, University of Twente, 2007.

[154] F. Siegemund. *Cooperating Smart Everyday Objects – Exploiting Heterogeneity and Pervasiveness in Smart Environments.* PhD thesis, ETH Zurich, 2004.

[155] M. Stäger, P. Lukowicz, and G. Tröster. Power and accuracy trade-offs in sound-based context recognition systems. *Pervasive and Mobile Computing*, 3(3):300–327, June 2007.

[156] Fred Stann and John Heidemann. RMST: Reliable data transport in sensor networks. In *Proceedings of the First International Workshop on Sensor Net Protocols and Applications*, pages 102–112, 2003.

[157] Thanos Stathopoulos, John Heidemann, and Deborah Estrin. A remote code update mechanism for wireless sensor networks. Technical report, UCLA, 2003.

[158] T. Stiefmeier, C. Lombriser, D. Roggen, H. Junker, G. Ogris, and G. Tröster. Event-Based Activity Tracking in Work Environments. In *International Forum on Applied Wearable Computing (IFAWC)*, March 2006.

[159] Martin Strohbach, Hans-Werner Gellersen, Gerd Kortuem, and Christian Kray. Cooperative artefacts: Assessing real world situations with embedded technology. In *Ubicomp*, pages 250–267, 2004.

[160] T. Terada, M. Tsukamoto, K. Hayakawa, T. Yoshihisa, Y. Kishino, A. Kashitani, and S. Nishio. Ubiquitous chip: A rule-based i/o control device for ubiquitous computing. In *Pervasive*, pages 238–253, 2004.

[161] D. Titterton and J. Weston. *Strapdown Inertial Navigation Technology, 2nd Edition*. Institution of Electrical Engineers, 2004.

[162] D. Towsley. An analysis of a point-to-multipoint channel using a go-back-n error control protocol. *IEEE Transactions on Communications*, 33(3):282–285, 1985.

[163] H. Tsai, O. Tonguz, C. Saraydar, T. Talty, M. Ames, and A. Macdonald. Zigbee-based intra-car wireless sensor networks: A case study. *IEEE Wireless Communications*, 14(6):67–77, 2007.

[164] K. Tuck. Tilt sensing using linear accelerometers. Technical Report AN3461, Freescale Semiconductor, 2007.

[165] E. van Stijn. *Miracle or Mirage? An exploration of the pervasive ERP phenomenon informed by the notion of conflicting memories*. PhD thesis, University of Twente, 2006.

[166] E. Waarts, Y. van Everdingen, and J. van Hillegersberg. The dynamics of factors affecting the adoption of innovations. *Journal of Product Innovation Management*, 19(6):412–423, 2002.

[167] Chieh-Yih Wan, Andrew T. Campbell, and Lakshman Krishnamurthy. PSFQ: A reliable transport protocol for wireless sensor networks. In *WSNA '02*, pages 1–11. ACM Press, 2002.

[168] Chieh-Yih Wan, Shane B. Eisenman, and Andrew T. Campbell. CODA: Congestion detection and avoidance in sensor networks. In *SenSys '03: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, pages 266–279, New York, NY, USA, 2003. ACM Press.

[169] T. Wang, Y. Han, P. Varshney, and P. Chen. Distributed fault-tolerant classification in wireless sensor networks. *IEEE J Sel Area Comm*, 23(4):724–734, April 2005.

[170] Z. Wang, P. Chen, Z. Song, Y. Chen, and K.L. Moore. Formation control in mobile actuator/sensor networks. *Unmanned Ground Vehicle Technology VII*, 5804(1):706–717, 2005.

[171] Z. Wang, Z. Song, P.-Y. Chen, A. Arora, D. Stormont, and Y. Chen. Masmote - a mobility node for mas-net (mobile actuator sensor networks). *Robotics and Biomimetics*, pages 816–821, 22-26 Aug. 2004.

[172] Wikipedia. Automotive industry. http://en.wikipedia.org/wiki/Automotive_industry.

[173] Wikipedia. Business rules. http://en.wikipedia.org/wiki/Business_rules.

[174] Wikipedia. Logistics. http://en.wikipedia.org/wiki/Logistics.

[175] Wikipedia. SCADA. http://en.wikipedia.org/wiki/SCADA.

[176] Wikipedia. VANET. http://en.wikipedia.org/wiki/VANET.

[177] A. Willig and H. Karl. Data transport reliability in wireless sensor networks. a survey of issues and solutions. *Praxis der Informationsverarbeitung und Kommunikation*, 28(2):86–92, 2005.

[178] A. Willig, K. Matheus, and A. Wolisz. Wireless technology in industrial networks. *Proceedings of the IEEE*, 93(6):1130–1151, 2005.

[179] A. Willig and R. Mitschke. Results of bit error measurements with sensor nodes and casuistic consequences for design of energy-efficient error control schemes. In *EWSN*, pages 310–325, 2006.

[180] D. Woods and T. Mattern. *Enterprise SOA Designing IT for Business Innovation*. O'Reilly, 2006.

[181] C. R. Wren, D. C. Minnen, and S. G. Rao. Similarity-based analysis for large networks of ultra-low resolution sensors. *Pattern Recogn.*, 39(10):1918–1931, 2006.

[182] J. Wu, S. Dulman, and P. Havinga. Reliable splitted multipath routing for wireless sensor networks. In *Proceedings of IFIP International Conference on Network and Parallel Computing*, pages 592–600, 2004.

[183] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks.

[184] L. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[185] L. Zadeh. The concept of a linguistic variable and its application to approximate reasoning. *Information Sciences*, 8:199 – 249, 1975.

[186] L. Zadeh. A computational approach to fuzzy quantifiers in natural languages. *Computers and Mathematics*, 9:149–184, 1983.

[187] Y. Zhang, S. Chatterjea, and P. Havinga. Experiences with implementing a distributed and self-organizing scheduling algorithm for energy-efficient data gathering on a real-life sensor network platform. In *International Workshop on From Theory to Practice in Wireless Sensor Networks*, 2007.

Publications

- M. Marin-Perianu, C. Lombriser, O. Amft, P. Havinga, G. Tröster. Distributed Activity Recognition with Fuzzy-Enabled Wireless Sensor Networks. In *International Conference on Distributed Computing in Sensor Systems (DCOSS)*, June 2008, Santorini Island, Greece.
- R. Marin-Perianu, C. Lombriser, M. Marin-Perianu, P. Havinga, G. Tröster. Towards Activity Recognition in Service-Oriented Body Area Networks. In *ICT Mobile and Wireless Communications Summit*, June 2008, Stockholm, Sweden.
- M. Horsman, M. Marin-Perianu, P. Jansen, P. Havinga. A Simulation Framework for Evaluating Complete Reprogramming Solutions in Wireless Sensor Networks. In *3rd International Symposium on Wireless Pervasive Computing (ISWPC)*, May 2008, Santorini Island, Greece.
- M. Marin-Perianu, N. Meratnia, P. Havinga, L. M. Sa De Souza, J. Muller, P. Spiess, S. Haller, T. Riedel, C. Decker, G. Stromberg. Decentralized Enterprise Systems: A Multiplatform Wireless Sensor Network Approach. *IEEE Wireless Communications Magazine*, Vol. 14(6), December 2007.
- C. Lombriser, M. Marin-Perianu, R. Marin-Perianu, D. Roggen, P. Havinga, G. Tröster. Organizing Context Information Processing in Dynamic Wireless Sensor Networks. In *3rd International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, December 2007, Melbourne, Australia.
- M. Marin-Perianu, P. Havinga. D-FLER: A Distributed Fuzzy Logic Engine for Rule-based Wireless Sensor Networks. In *4th International Symposium on Ubiquitous Computing Systems (UCS)*, November 2007, Tokyo, Japan.

- R. Kauw-A-Tjoe, J. Thalen, M. Marin-Perianu, P. Havinga. SensorShoe: Mobile Gait Analysis for Parkinson's Disease Patients. In *UbiWell Workshop*, colocated with Ubicomp, September 2007, Innsbruck, Austria.
- R. Marin-Perianu, M. Marin-Perianu, P. Havinga, J. Scholten. Movement-based Group Awareness with Wireless Sensor Networks. In *5th International Conference on Pervasive Computing (Pervasive)*, May 2007, Toronto, Canada.
- M. Marin-Perianu, P. Havinga. RMD: Reliable Multicast Data Dissemination within Groups of Collaborating Objects. In *First IEEE International Workshop on Practical Issues in Building Sensor Network Applications*, colocated with LCN, November 2006, Tampa, USA.
- M. Marin-Perianu, T. Hofmeijer, P. Havinga. Implementing Business Rules on Sensor Nodes. In *11th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, September 2006, Prague, Czech Republic.
- M. Marin-Perianu, N. Meratnia, M. Lijding, P. Havinga. Being Aware in Wireless Sensor Networks. In *15th IST Mobile & Wireless Communication Summit, Capturing Context and Context Aware Systems and Platforms Workshop*, June 2006, Myconos, Greece.
- M. Marin-Perianu, T. Hofmeijer, P. Havinga. Assisting Business Processes Through Wireless Sensor Networks. In *13th International Conference on Telecommunications (ICT)*, May 2006, Madeira Island, Portugal.
- M. Marin-Perianu, P. Havinga. Experiments with Reliable Data Delivery in Wireless Sensor Networks. In *Intelligent Sensors, Sensor Networks and Information Processing Conference (ISSNIP)*, pages 169-174, December 2005, Melbourne, Australia.
- L. Evers, M. Bijl, M. Marin-Perianu, R. Marin-Perianu, P. Havinga. Wireless Sensor Networks and Beyond: A Case Study on Transport and Logistics. In *International Workshop on Wireless Ad-hoc Networks (IWWAN)*, May 2005, London, UK.