Wireless Sensor Networks in Motion

Clustering Algorithms for Service Discovery and Provisioning

Raluca Marin-Perianu

The work in this thesis has been supported by PROGRESS, the embedded systems research programme of the Dutch organisation for Scientific Research NWO, the Dutch Ministry of Economic Affairs and the Technology Foundation STW. The research has been carried out within the context of the Center for Telematics and Information Technology (CTIT) and under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

Keywords: wireless sensor networks, service discovery protocols, clustering algorithms, context awareness, movement detection

Cover Design: Newblack, `www.newblack.ro`.
Printed by Wöhrmann Print Service.

IPA Dissertation Series No. 2008-29
CTIT PhD Thesis Series No. 08-130
ISBN 978-90-365-2745-3

WIRELESS SENSOR NETWORKS IN MOTION

CLUSTERING ALGORITHMS FOR SERVICE DISCOVERY AND PROVISIONING

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 13.15

by

Raluca Sandra Marin-Perianu

born on the 3rd of October 1978

in Bucharest, Romania

This dissertation is approved by:

Prof. Dr. P.H. Hartel (promotor)

Abstract

The evolution of computer technology follows a trajectory of miniaturization and diversification. The technology has developed from mainframes (large computers used by many people) to personal computers (one computer per person) and recently, embedded computers (many computers per person). One of the smallest embedded computers is a wireless sensor node, which is a battery-powered miniaturized device equipped with processing capabilities, memory, wireless communication and sensors that can sense the physical parameters of the environment. A collection of sensor nodes that communicate through the wireless interface form a Wireless Sensor Network (WSN), which is an ad-hoc, self organizing network that can function unattended for long periods of time.

Although traditionally WSNs have been regarded as static sensor arrays used mainly for environmental monitoring, recently, WSN applications have undergone a paradigm shift from static to more dynamic environments, where nodes are attached to moving objects, people or animals. Applications that use WSNs in motion are broad, ranging from transport and logistics to animal monitoring, health care and military, just to mention a few.

These application domains have a number of characteristics that challenge the algorithmic design of WSNs. Firstly, mobility has a negative effect on the quality of the wireless communication and the performance of networking protocols. Nevertheless, it has been shown that mobility can enhance the functionality of the network by exploiting the movement patterns of mobile objects. Secondly, the heterogeneity of devices in a WSN has to be taken into account for increasing the network performance and lifetime. Thirdly, the WSN services should ideally assist the user in an unobtrusive and transparent way. Fourthly, energy-efficiency and scalability are of primary importance to prevent the network performance degradation.

This thesis focuses on the problems and enhancements brought in by network mobility, while also accounting for heterogeneity, transparency, energy-

efficiency and scalability. We propose a set of algorithms that enable WSNs to self-organize efficiently in the presence of mobility, adapt to and even exploit dynamics to increase the functionality of the network. Our contributions include an algorithm for motion detection, a set of clustering algorithms that can be used to handle mobility efficiently, and a service discovery protocol that enables dynamic user access to the WSN functionality. In short, the main contributions of this thesis are the following:

1. **Classifications of service discovery protocols and clustering algorithms.** We systematically analyse the discovery and clustering mechanisms for WSNs through a thorough review and classification of the state of the art.

2. **A generalized clustering algorithm for wireless sensor networks.** We propose a clustering algorithm for dynamic sensor networks, which represents a generalization of a set of state-of-the-art clustering algorithms. This generalized algorithm allows for a better understanding of the specialized algorithms and facilitates the definition and demonstration of common properties.

3. **Cluster-based service discovery for wireless sensor networks.** We propose a cluster-based service discovery solution for heterogeneous and dynamic wireless sensor networks. The service discovery protocol exploits a cluster overlay for distributing the tasks according to the capabilities of the nodes while providing an energy-efficient search. The clustering algorithm is designed to function as a distributed service registry.

4. **On-line recognition of joint movement in wireless sensor networks.** We propose a method through which dynamic sensor nodes determine whether they move together by communicating and correlating their movement information. The movement information is acquired from tilt switches and accelerometer sensors.

5. **A context-aware method for spontaneous clustering of dynamic wireless sensor nodes.** We propose a clustering algorithm that organizes wireless sensor nodes spontaneously and transparently into clusters based on a common context, such as movement information.

Through these contributions, the thesis opens novel perspectives for WSN applications in the field of distributed situation assessment, where sensor nodes can collaboratively determine the movement characteristics of the people or moving objects and organize in structures that correspond to the real world.

Samenvatting

Computertechnologie is door de jaren heen in de richting van miniaturisatie en diversificatie geëvolueerd. Deze technologie heeft zich ontwikkeld van mainframes (grote computers gebruikt door vele mensen tegelijk) naar personal computers (slechts één computer per persoon) en recentelijk naar embedded computers (vele computers per persoon). Een van de kleinste embedded computers is de wireless sensor node. Dit is een geminiaturiseerd apparaatje dat op een batterij werkt en voorzien is van een processor, geheugen, draadloze communicatie en sensoren die fysieke eigenschappen van de omgeving kunnen waarnemen. Een groep van deze kleine apparaatjes vormt een draadloos sensornetwerk (Wireless Sensor Network, WSN) door middel van onderlinge draadloze communicatie. Dit is een ad hoc netwerk dat zichzelf inricht en als zodanig zonder begeleiding voor langere tijd kan functioneren.

Traditioneel werden WSNs beschouwd als statische sensoropstellingen bedoeld voor het doen van milieumetingen. Echter zijn er recentelijk ook WSN toepassingen ontstaan voor meer dynamische omgevingen, waarbij de sensor nodes zijn bevestigd aan bewegende objecten, mensen of dieren. Een breed spectrum aan toepassingen voor dergelijke beweeglijke sensornetwerken is denkbaar, variërend van transport en logistiek tot bijvoorbeeld het observeren van dieren, de gezondheidszorg en defensie.

Deze toepassingsgebieden hebben een aantal eigenschappen die het ontwikkelen van algoritmen voor WSNs bemoeilijken. Ten eerste heeft mobiliteit een negatief effect op de kwaliteit van de draadloze communicatie en de prestatie van de gebruikte netwerkprotocollen. Niettemin is aangetoond dat mobiliteit de functionaliteit van het netwerk kan verbeteren door de bewegingspatronen van de mobiele objecten slim te gebruiken. Ten tweede moet rekening gehouden worden met de heterogene eigenschappen van de apparaten in het WSN om de prestaties en de levensduur van het netwerk te verbeteren. Ten derde zouden de diensten die het WSN aanbiedt de gebruiker op een onopval-

lende en transparante wijze moeten assisteren in zijn bezigheden. Ten vierde zijn het efficiënt gebruik van energie en de schaalbaarheid van het systeem van het grootste belang om te voorkomen dat de prestaties van het netwerk gaandeweg afnemen.

Dit proefschrift behandelt de problemen en verbeteringen die voortvloeien uit de mobiliteit van het netwerk, terwijl er ook rekening gehouden wordt met heterogene eigenschappen van de nodes, transparantheid van de diensten, schaalbaarheid van het netwerk en efficiënt gebruik van energie. Wij presenteren algoritmen voor WSNs die het netwerk de mogelijkheid geven zichzelf efficiënt in te richten in mobiele toepassingen. Met deze algoritmen kunnen WSNs zich aanpassen aan of zelfs gebruik maken van de dynamiek van de omgeving om de functionaliteit van het netwerk te vergroten. Onze bijdragen omvatten onder andere een algoritme voor het detecteren van beweging, een verzameling cluster-algoritmen die gebruikt kunnen worden om mobililiteit efficiënt af te handelen en een service discovery protocol dat de gebruiker dynamisch toegang verleent tot de functionaliteit van het WSN. Samenvattend zijn de bijdragen van dit proefschrift als volgt:

1. **Classificatie van service discovery protocollen en clustering algoritmen.** We geven een systematische analyse van discovery en clustering mechanismen voor WSNs door middel van een uitgebreide evaluatie en classificatie van de huidige technologie.

2. **Een gegeneraliseerd clustering algoritme voor draadloze sensornetwerken.** We presenteren een clustering algoritme voor dynamische sensor netwerken. Dit algoritme is een generalisatie van bepaalde clustering algoritmen uit de huidige stand van de technologie. Door deze generalisatie voorziet dit algoritme in de mogelijkheid algoritmen die zijn ontworpen in mobiele WSN omgevingen beter te doorgronden. Tevens maakt dit het definiëren en demonstreren van gemeenschappelijke eigenschappen van dergelijke algoritmen eenvoudiger.

3. **Een service discovery mechanisme voor draadloze sensornetwerken dat gebaseerd is op clustering.** We geven een gecombineerde en op clusters gebaseerde service discovery oplossing voor heterogene en dynamische sensornetwerken. Het service discovery protocol maakt slim gebruik van clustering om de taken over het netwerk te verdelen afhankelijk van de capaciteiten van de individuele nodes. Dit protocol maakt efficiënt gebruik van energie bij het uitvoeren van zoekopdrachten. Het clustering

algoritme is zodanig van opzet dat het functioneert als een gedistribueerd register van alle services die in het netwerk aangeboden worden.

4. **Het herkennen van gemeenschappelijke bewegingen in draadloze sensornetwerken terwijl het systeem actief is.** We geven een methode waarmee dynamische sensor nodes kunnen bepalen of zij gezamenlijk bewegen. Dit wordt gedaan door het communiceren en correleren van bewegingsinformatie. Deze informatie wordt binnen de nodes verkregen uit metingen van hellings- en acceleratiesensoren.

5. **Een context-aware methode voor het spontaan vormen van clusters met dynamische wireless sensor nodes.** We presenteren een methode waarmee wireless sensor nodes zichzelf spontaan en transparant in clusters kunnen organiseren door het herkennen van een gemeenschappelijke context. Dit kan bijvoorbeeld gemeenschappelijke beweging betreffen.

Door middel van deze bijdragen biedt dit proefschrift nieuwe mogelijkheden voor gedistribueerde situatiebeoordeling, waar samenwerkende sensor nodes de bewegingseigenschappen van mensen of objecten kunnen bepalen en zich vervolgens kunnen organiseren in structuren die overeenstemmen met structuren in de echte wereld.

Acknowledgements

The last four years have been the most beautiful of my life. A job that I liked, people that I met and so many trips make me feel that I made a wise decision in the summer of 2004, when I accepted a PhD position at University of Twente.

I would like to thank my promoter, Pieter Hartel, for coordinating my research during these four years. I learned a lot from Pieter about the proper way of conducting research, how to produce scientific solid results and how to structure and present the work. We had many discussions, sometimes with divergence of opinions, when I learned how to use proper argumentation, to construct precise statements and to have a strong motivation for each decision step. I believe that without Pieter the thesis would not reach so far.

I thank Hans Scholten for being my daily coach. We had so many interesting discussions, not only about research, but also about our leisure time. Hans even lent me his own notebook when mine crashed and I was desperately in need of one to finish up the thesis writing.

Paul Havinga was the project leader and he had a major influence on my research. All the discussions with him were very inspiring and he is a great source of novel ideas. He is also a very enthusiastic person, who knows how to motivate people. Paul helped me and Mihai to go for an internship at ETH Zürich, which was a very useful experience. I thank Gerhard Tröster for welcoming us at ETH and coordinating our research there. We worked in close collaboration with Clemens Lombriser, who is a very good co-author and friend. The results from this collaboration represent an important chapter of my thesis.

I would like to thank Tim Nieberg and Johann Hurink for their help with mathematical proofs. Mathematicians proved eminently precise in their statements, while being at the same time friendly and open-minded persons.

I also wish to thank Nirvana Meratnia for being next to me at the most difficult moments during my PhD studentship. Her sound judgement has helped me a lot to overcome the periods of disappointment and confusion.

I thank Hans Gellersen, Christian Bettstetter, Ed Brinksma, Boudewijn Haverkort and Johann Hurink for assessing my manuscript as members of my graduation committee. I would also like to acknowledge Frank Karelse, Arnold Ardenne, Jo De Boeck, Eelco Dijkstra, Maarten Ditzel, Wim Hendriksen, Harry Kip, Kees Nieuwenhuis and John de Waal for their interesting comments and discussions about my research during the Featherlight Project meetings. Many thanks to our secretaries Marlous, Nicole and Thelma for constantly helping us with the annoying administrative issues.

Our friends from the university, Oana and Vali, came to the Netherlands one year earlier than us. They described the life in the Netherlands and research conditions in such positive terms, that we were finally convinced to apply for PhD positions. We found out that what they told us was accurate, so I thank Oana and Vali for being our guides to this foreign country.

Ileana and Stefan are the best friends that we have made during these four years, and we spent so much time together in Enschede, all over Europe and even Australia, that now it feels weird that we have to start living in different towns. Andreea and Eugen made our small Romanian group even more charming and entertaining. However, nothing is as good as the lively and "gezellig" atmosphere of our large parties and volleyball sessions, where the Turkish Mafia is always present, accompanied, of course, by Michel, our Dutch friend. Our Turkish friends grew in number as time passed by, namely Özlem, Mustafa, Seçkin, Cem, Ayşegül, Kamil, Ayşe, Murat. But our list of friends is much larger and even more international. We had so many nice moments with Sinan, Anka, Ari, Ha, Roland, Chiara, Supriyo, Anindita, Blas, Kavitha, Kiran, Hugo, Yang, Malohat, Maria, Mohamed, Lodewijk, Laura, Ricardo, Vasughi, Luminita, Diana, Georgiana. We wrote project proposals with Supriyo and Nirvana, learned salsa steps from Blas, took Dutch lessons from Lodewijk, Tjerk and Michel, drove sensor-enabled Ferrari cars together with Stephan (many thanks for an excellent Samenvatting!), but truly unique was our girl dancing team! Together with Nirvana, Ileana, Özlem and Kavitha as our teacher, we learned and performed synchronous Bollywood Indian dancing, which was the highlight of many parties and late office hours.

However, nothing would have been possible without the committed support of my family - my parents, grandmother, uncles, parents-in-law and godparents - thank you for being next to us all these years! I also thank Irina and Andrei for the cover design, as well as for their constant help in designing project logos and figures with application settings.

For Mihai, my other half, I simply cannot express in words my gratitude for sharing with me each moment of our lives.

Contents

Chapter 1

Introduction

The ubiquitous computing vision [164] defined by Marc Weiser in 1991 describes the computer of the future as an invisible technology completely integrated into our environment. The user will be surrounded by unnoticeable and omnipresent computers and will use them unconsciously to accomplish everyday tasks. Wireless Sensor Networks (WSNs) represent an enabling technology [71, 31] that contributes to the realization of this vision. This chapter presents the WSN technology and introduces the most relevant applications.

1.1 Wireless sensor networks

A wireless sensor *node* consists of a microcontroller, a radio, several sensors, storage and a battery. A WSN is composed of sensor nodes that sense several environmental phenomena and form an ad-hoc network for the purpose of collaboratively processing and transmitting the data to the interested parties. A WSN is a *self-organizing network* that does not need user intervention for configuration or setting up routing paths. Therefore, WSNs can be used in virtually any environment, even in inhospitable terrain or where the physical placement is difficult [144].

The traditional WSN application is environmental monitoring, where static sensor arrays are deployed to collect sensor readings from large or remote geographical areas to a *central point* (or *base station, sink*) [117]. Therefore, algorithmic research in WSN has mostly focused on the study and design of *energy-efficient and scalable* algorithms for data transmission from the sensor nodes to the base station. Recently, the WSN applications have undergone a paradigm shift from static to more dynamic environments, where nodes are mobile, as they are attached to people, animals and moving objects (see Section 1.1.3). Consequently, algorithmic research in WSN has to move from static data collection to a more dynamic concept, which represents the focus of this thesis.

In what follows, we present a survey of the current WSN hardware and software technology, together with the evolution of WSN application scenarios.

1.1.1 Hardware

To have an image of the current WSN technology, we present in Table 1.1 a survey of the commercially available wireless sensor platforms, ordered by the type of radio. By analysing Table 1.1, we notice the following characteristics of sensor nodes:

- *Heterogeneity.* Today's WSN market is heterogeneous and offers an entire spectrum of sensor nodes, ranging from small devices with limited hardware resources, such as the Ambient SmartTag [13], to powerful nodes approaching the capabilities of an embedded computer, such as IMote2 [101].

- *Interoperability.* Initially developed with proprietary wireless networking protocols operating in the 868/915 MHz band, the WSN market converges towards a uniformly accepted network standard, with IEEE 802.15.4 [17] and ZigBee [27] being the prominent options. As a consequence of these standardization efforts, WSN platforms are expected to become interoperable.

Heterogeneity and interoperability have the potential to expand the WSN functionality and to increase the quality of service, by putting together the flexibility of resource-lean nodes and the enhanced capabilities of more endowed nodes [54, 120].

1.1.2 Software

In what follows, we review the main directions of research in the software field of WSNs, covering the algorithms and protocols needed to achieve the application requirements and functionality:

Platform	**Radio**	**Processor**	**RAM**	**Flash**	**Sensors/Actuators**
Ambient μNode [13]	868/915 MHz	8 MHz TI MSP430	10kB	48kB	I/O ports, 3 LEDs, LCD
Ambient SmartTag [13]	868/915 MHz	16 MHz Intel 8051	128 bytes	4kB	I/O ports, LED
Teco uPart [25]	868/915 MHz	4 MHz PIC12F675	64 byte	1.4kB	Movement, light sensor, temperature, LED
Crossbow MICA2 [15]	868/915 MHz	8 MHz Atmel ATmega128L	4kB	128kB	I/O ports, 3 LEDs
Crossbow MICAz [15]	2.4 GHz IEEE 802.15.4	8 MHz Atmel ATmega128L	4kB	128kB	I/O ports, 3 LEDs
Crossbow IMote2 [15]	2.4 GHz IEEE 802.15.4	13-416 MHz PXA271 XScale	32MB	32MB	I/O ports, camera interface, LED
Crossbow Iris [15]	2.4 GHz IEEE 802.15.4	Atmel ATmega1281	8kB	128kB	I/O ports, 3 LEDs
Crossbow TelosB [15]	2.4 GHz IEEE 802.15.4	8 MHz TI MSP430	10kB	48kB	Light, temperature, humidity
SensiNode NanoSensor N710 [22]	2.4 GHz IEEE 802.15.4	32 MHz TI CC2430	8kB	128kB	Temperature, light, 2 LEDs
SensiNode Micro.2420 [22]	2.4 GHz IEEE 802.15.4	8 MHz TI MSP430	10kB	256kB	Stackable
Sun SPOT [20]	2.4 GHz IEEE 802.15.4	180MHz ARM920T	512kB	4MB	3-axis accelerometer, temperature, light, I/O ports, 8 LEDs
Sentilla Tmote Sky [23]	2.4 GHz IEEE 802.15.4	8Mhz TI MSP430	10kB	48kB	Light, I/O ports
Sentilla Tmote Mini [23]	2.4 GHz IEEE 802.15.4	8Mhz TI MSP430	10kB	48kB	I/O ports
XYZ [26]	2.4 GHz IEEE 802.15.4	1-60 MHz OKI Semiconductor ML67 ARM	32kB	256kB	Accelerometer, temperature, light, I/O ports

Table 1.1: Wireless sensor platforms

- *Operating systems.* Operating systems for WSNs are designed to manage the sensor node resources and provide programmers with an interface to access these resources [111, 88]. Operating systems for WSNs are typically less complex than general-purpose operating systems, mainly because of the resource constraints of the hardware platforms. For example, they do not include support for user interfaces or provide virtual memory techniques.

- *Networking protocols.* Due to the embedded nature of wireless sensor nodes, the need for self-organisation, energy efficiency, scalability and robustness, a new breed of networking protocols has been designed for WSNs. Medium Access Control (MAC) protocols must be power-aware and able to use the wireless channel efficiently by avoiding collisions and minimizing delay [87]. The network layer takes care of routing the data from source to destination (typically the sink node) in an energy-efficient manner. The transport layer is responsible for congestion control and reliable data delivery [118].

- *Specific protocols.* To be able to improve the performance of networking protocols and to meet the application requirements, specific protocols have been designed, such as clustering [83], localization [67], security [106] and data acquisition, manipulation and storage [116].

- *High-level dissemination.* The interaction between the WSN and the outside world is done via a high-level dissemination layer, where the functionality of the WSN is offered in a uniform way to the user [65, 120].

As pointed out by Tanenbaum et al. [154], the WSN software design process should also consider the *system aspects*, in order to deliver the expected functionality from the application perspective. Therefore, this research is conducted such that the designed algorithms are implemented and tested on real sensor nodes (see Section 1.3). In what follows, we provide an overview of the main application domains of WSNs and identify the related system characteristics and technological challenges, which subsequently lead us to the research questions and contributions of this thesis.

1.1.3 Applications

The range of WSNs applications has extended considerably, mainly due to the following reasons: (1) the processing capabilities of the nodes have evolved up to a point that enables them to execute complex tasks and to make decisions autonomously; (2) a group of sensor nodes can combine their resources and capabilities through collaboration and provide complex services, such as reliable event detection, localization or tracking; and (3) an interoperable collection of heterogeneous devices can achieve superior functionality by using the flexibility of the resource-lean devices in conjunction with the enhanced capabilities of the more endowed nodes [54]. Therefore, the functionality of WSNs evolves from the traditional data gathering to more complex applications, as shown by the following succinct review:

Environmental monitoring. Environmental monitoring is the traditional WSN application, where a static array of sensors is randomly or uniformly deployed over an area to gather sensor readings and to transmit them at a central point for processing. Typical settings include precision agriculture [38], habitat monitoring [117] or ocean water monitoring [57].

Animal monitoring. Different from the environmental monitoring applications by introducing mobility within WSNs, this scenario assumes that nodes are attached to animals for the purpose of studying their behaviour, locating or confining them within an area. Examples include wild life monitoring [95] and cattle herding [46].

Health care. Sensor nodes integrated into garments, also known as Body Area Networks (BANs), can be used to monitor the vital signs of patients [40], their walking pattern [98], or even to localize the patients or medical personnel inside a building [37].

Industrial safety. Industrial safety can benefit from the WSN technology by verifying in real-time the safety regulations at industrial sites. Sensor nodes can collaboratively determine and prevent potential hazardous situations, and alert or take action at the point of interest. For example, in the oil and gas industry, dangerous situations may arise by storing incompatible substances in close proximity of each other or exceeding the maximum storage volume threshold for hazardous substances [120].

Smart buildings. Sensor networks can provide monitoring and control of environmental conditions in buildings (such as temperature, humidity, or light), electronic door and way signs [113], localization of people [97].

Emergency. Emergency applications have as main objective the rescue of people in danger. For this purpose, people at risk carry a sensor node that permits localization in case of disaster. Example applications include rescue of avalanche victims [126] and fire fighting and rescue [142].

Military. WSN represents a promising technology for military applications because low-cost, disposable sensor nodes can be deployed in these destructive environments. Some of the military applications of sensor networks are: battlefield surveillance, mapping opposing terrain, nuclear, biological and chemical attack detection and reconnaissance, target tracking [12].

Transport and logistics. Transport and logistics represent an important market for WSNs. The goal is to monitor the storage conditions of products, to verify the loads, and to real-time localize the goods at production sites, distribution centres or stores [1].

As this particular application domain has been a valuable inspiration point for this thesis (see Chapters 6 and 7), we give in the following a detailed description of the transport and logistics processes, highlighting the typical errors involved and the role of the WSN to deal with these errors and to improve efficiency.

The process starts at a *warehouse*, where an *order picker* gets an order list and assembles a *Returnable Transport Item (RTI, rolling container or cart)*, picks the requested products from the warehouse shelves and loads them in the RTI. Once the RTI is full or the order is complete, the order picker puts a sticker with a barcode or RFID on the RTI, which henceforth uniquely identifies this RTI. Then, the RTI is moved to the *expedition floor*, a large area used for temporary storage. A grid is painted on the expedition floor and each cell of the grid is associated with a certain shop. Loaded RTIs arrive on the expedition floor and are placed depending on the shop they are assigned to. The RTIs belonging to one shop are grouped together and occupy one or more adjacent grid cells, depending on the order size. At loading time, the *loading operators* place the RTIs into trailers, according to a *loading list*, derived from the delivery orders. Eventually, a truck pulls the trailer and delivers the goods to the shops. Upon arrival at a store, some or all the RTIs are unloaded from the trailer. If available, previously delivered dismantled RTIs are loaded into the trailer, to be returned to the distribution centre and reused in a future delivery.

Keeping track of the status of a certain order is currently carried out by means of barcode or RFID scanning. The scanning occurs at several stages:

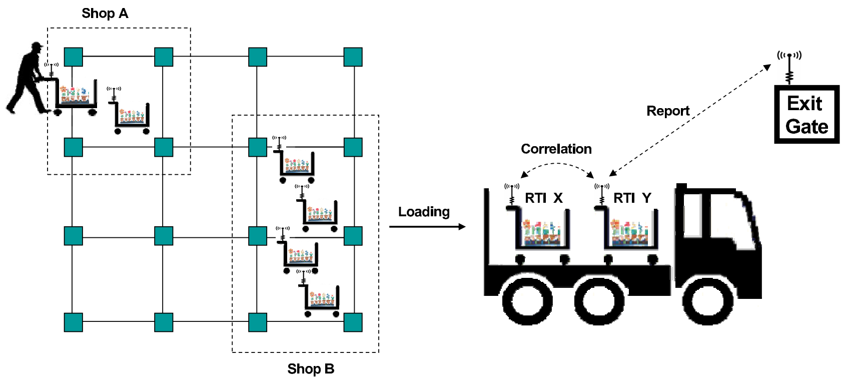


Figure 1.1: Transport and logistics process diagram: placement of RTIs on the expedition floor and loading RTIs into trailers.

when assembling an RTI and associating it with an order, when verifying completeness and loading sequence of an order, etc. However, due to the large scale of the process, the transport company personnel (e.g. order pickers, loading operators) make errors. It often happens that the order pickers make mistakes when filling the RTIs with goods, or that the RTIs are placed in a wrong cell or lost on the expedition floor, loaded in the wrong trailer or not returned from retail stores. In addition, the products are sometimes stored in improper climate conditions, which is a serious problem in the case of perishable goods.

The conclusion is that many of the current problems occur as a result of incorrect handling and storage of products and RTIs at various stages of the distribution process. The process efficiency can be improved by using the WSN technology: attaching sensor nodes to products and RTIs and also deploying them as a fixed infrastructure. This will ensure the reduction or removal of the most common causes of errors currently experienced. Figure 1.1 shows the transport and logistics process diagram, where a fixed infrastructure of sensor nodes is placed uniformly in the grid on the expedition floor. These nodes are referred to as *beacons* and facilitate the localization process of the RTIs on the expedition floor. Groups of RTIs placed within adjacent cells are then transported together in the same trailer. Each RTI is equipped with a wireless sensor node, termed a *micronode*, while a *piconode* is attached to each product. The nodes are equipped with sensors for sampling temperature, humidity, light and other environmental conditions, and also movement sensors, which can be used for verifying the loading of products in RTIs and of RTIs into trailers (see Chapter 6 for a detailed description of the verification process). Sensors can also be attached to order pickers or loading operators, such that the loading process can be automatically verified and the transport company personnel can be localized whenever necessary.

To summarize, WSN technology can bring the following improvements to the current transport and logistics processes:

- Automatic verification of loads (products loaded into RTIs and RTIs loaded into trailers).
- Discovery and localization of products and RTIs in the warehouse, expedition floor and shops.
- Discovery and localization of transport company personnel.
- Verification of environmental and storage conditions.

The WSN environment for transport and logistics applications is *dynamic*, with both mobile and static nodes, and *heterogeneous*, with beacons, micronodes and piconodes wirelessly interacting to improve the efficiency of the process. Dynamics and heterogeneity are in fact more general system properties, common to most of the application domains previously discussed. This observation indicates that WSNs evolve beyond the static sensor array model towards interactive mobile nodes attached to people, animals, objects, and from the homogeneous Smart Dust vision [163] to resource-aware, heterogeneous nodes, specialized on specific tasks according to the application design. Building on these generic system aspects, we outline in the following the major challenges of WSNs in achieving the ubiquitous computing vision.

1.1.4 Challenges

The functionality of a WSN is dependent on the application domain. In the traditional monitoring applications, nodes are generally static after deployment and their role is limited to data collection and gathering to a central point for processing [144]. Changes in the network topology are infrequent and thus WSN protocols generally assume a static data collection pattern [83]. Analysing the above broad spectrum of applications, we deduce that a growing number of WSNs are *dynamic* environments, where nodes change their position in real-time. Consequently, algorithms and protocols for self-organization in WSNs have to take into account mobility from the design phase. In this way, they can

(1) reduce the negative impact of mobility on the performance of networking protocols, and (2) exploit the potential of using mobility to enhance the WSN functionality.

To conclude, we summarize in the following the major challenges that WSNs face in order to contribute to the ubiquitous computing vision [103, 62]:

- **Heterogeneity**. Collaboration among sensor nodes with different hardware capabilities offers more flexibility and supports elaborate tasks, but at the same time forces algorithms and protocols to become *resource-aware*. Therefore, resources in a WSN have to be discovered and efficiently managed for an improved network functionality and performance.

- **Dynamics**. The paradigm shift from static sensor arrays to pervasive applications involves a major increase in the overall degree of mobility or dynamics. Mobility thus becomes an *intrinsic system property*, which needs to be considered even from the protocol design phase. Although mobility has a negative effect on the quality of wireless communication [130] and the performance of networking protocols [72, 66], there are still cases when mobility turns out to be a means of enhancing network performance (e.g. data mule [145]) or a new way of solving a given problem (e.g. authentication through spontaneous interaction [123]).

- **Proactivity and transparency**. Proactive WSNs have the potential of delivering context-aware and just-in-time services. The challenge is to provide the user with "what I want" information and services in a transparent manner. As remarked by Kumar and Das [103], typical examples of proactive services currently available, such as the online paper clip and pop-up messages, are obtrusive and often useless. WSN-based proactive services should ideally assist the user in an unobtrusive way and at the same time ensure efficient utilization of resources.

In addition to this list, the traditional WSN challenges remain a continuous concern in the protocol design process. Firstly, since sensor nodes are battery-powered, **energy-efficiency** is of primary importance to assure a long network lifetime. Secondly, **scalability** with respect to the number and density of nodes is essential to prevent the degradation of network performance below the acceptable threshold.

1.2 Research question

In view of the above challenges, this thesis focuses on the *dynamics* of WSNs, while also accounting for heterogeneity, transparency and the traditional WSNs objectives defined in Section 1.1 (i.e. energy-efficiency and scalability). We formulate the main research question that this thesis addresses:

Research question *How can WSNs self-organize efficiently in presence of mobility, adapt to and even exploit dynamics to increase the functionality of the network?*

Self-organization and adaptation in dynamic WSNs involves multiple mechanisms at distinct levels of abstraction. We distinguish the following:

1. *Low-level networking.* WSNs have to implement the low-level networking primitives in a distributed fashion. More specifically, sensor nodes negotiate the access to the wireless medium, coordinate data packet routing and regulate error and congestion control.

2. *Clustering.* An overlay network topology can be used to handle mobility, either by selecting the least mobile nodes as part of the overlay, or by organizing the nodes according to their semantic relationship, such as moving together. Consequently, clustering can be used for the following specific purposes:

   (a) *Reducing the effect of mobility on networking protocols.* Mobility has a negative effect on networking protocols for sensor networks, inducing delays, message overhead or can even make protocols unoperational. Clustering can help reduce the effect of mobility on networking protocols, by making a highly dynamic topology appear less dynamic [124].

   (b) *Enhance the network functionality.* Spontaneous clustering based on similar mobility pattern of sensor nodes can be used to enhance the functionality of the network, by delivering contex-aware services such as activity recognition.

3. *Discovery.* To be able to access the WSN functionality within a dynamic environment, one must be able to *discover* the nodes, resources and services available at each moment in the sensor network. Therefore, providing a discovery mechanism is essential for pervasive computing applications.

4. *High-level distributed processing.* The goal of this layer is to have self-organizing WSNs that carry out tasks distributively, by sharing resources and providing services in a transparent way to the user. Consequently, this layer deals with problems such as distributed shared memory, dynamic task allocation and information fusion.

This thesis focuses on the middle tiers, i.e. items 2 and 3, as we describe in the contributions from the following section.

1.3 Contributions

With regard to the previously mentioned challenges and research question, we describe in the following the main contributions of the thesis. To clarify the relations between the research issues and our contributions, the reader is referred to Table 1.2.

Contribution 1 ***Classifications of service discovery protocols and clustering algorithms - Chapters 2 and 3***

To be able to analyse systematically the discovery and clustering mechanisms for pervasive environments, we review the state of the art service discovery protocols and clustering algorithms. In both cases, we follow three methodological steps: (1) define the problem, general objectives and properties, (2) classify the existing solutions with respect to the defined objectives and properties and (3) frame the state of the art in a comparative table according to the proposed classification.

Contribution 2 ***A generalized clustering algorithm for wireless sensor networks - Chapter 4***

We propose a generalized clustering algorithm for dynamic sensor networks, which allows for a better understanding of algorithms designed in mobile WSN environments, and facilitates the definition and demonstration of common properties for such algorithms. The description of the generalized algorithm is presented in the paper [2].

Contribution 3 ***Cluster-based service discovery for wireless sensor networks - Chapter 5***

We propose a combined, cluster-based service discovery solution for heterogeneous and dynamic wireless sensor networks. The service discovery protocol

Research issue	Contribution	Chapter
2-Clustering	1-Classification of clustering algorithms 2-Generalized clustering algorithm	3 4
2a-Clustering, reduce the effect of mobility	3-Cluster-based service discovery	5
2b-Clustering, enhance network functionality	4-Recognition of joint movement 5-Context-aware spontaneous clustering	6 7
3-Discovery	1-Classification of discovery protocols 3-Cluster-based service discovery	2 5

Table 1.2: Research issues and corresponding contributions of this thesis.

exploits a cluster overlay for distributing the tasks according to the capabilities of the nodes and providing an energy-efficient search. The clustering algorithm is explicitly designed to function as a distributed service registry and represents a particular case of the generalization proposed by Contribution 2. We analyse theoretically and through simulations how the properties of the clustering structure influence the performance of the service discovery protocol. The design and analysis of the proposed solution appears in papers [3] and [4]. To validate our results, we implement the proposed solution on resource-constraint sensor nodes and we measure the performance of the protocol running on different testbeds. The implementation details and experimental results are described in the paper [5]. We build a demonstration setting as a proof of concept of our combined solution, described in [10].

Contribution 4 ***On-line recognition of joint movement in wireless sensor networks - Chapter 6***

We propose a method through which dynamic sensor nodes determine that they move together by communicating and correlating their movement information. The final goal is to provide a clustering criteria based on semantic properties (joint movement). The movement information is acquired from tilt switches and accelerometer sensors. We implement a fast, incremental correlation algorithm, which can run on resource constrained devices. As recommended by Tanenbaum et al. [154] (see Section 1.1.2), we test the solution in real life: we attach sensors to RTIs and cars, as a direct application of the transport and logistics processes described in Section 1.1.3. Computations are done online and the results show that the method distinguishes between joint and separate movements. The solution using tilt switches proves to be simpler, cheaper and

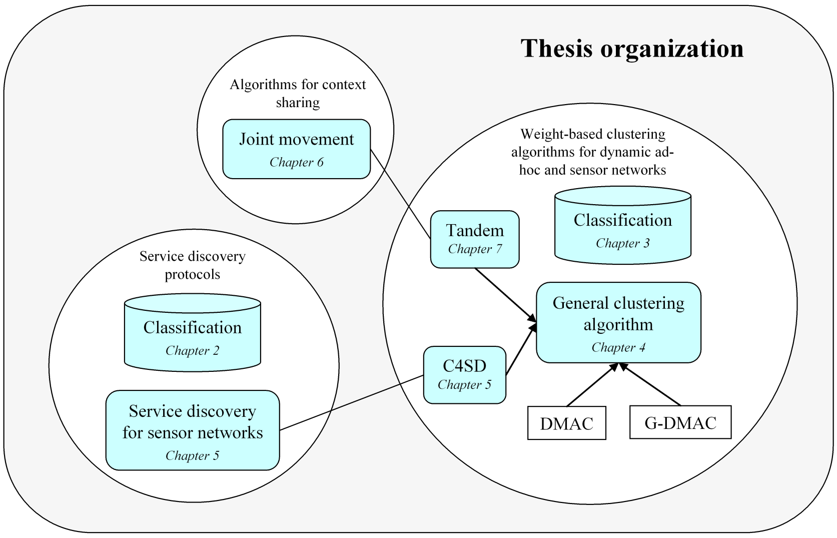

Thesis organization
Algorithms for context sharing
Joint movement
Chapter 6
Weight-based clustering algorithms for dynamic ad-hoc and sensor networks
Classification
Chapter 3
Tandem
Chapter 7
Service discovery protocols
Classification
Chapter 2
General clustering algorithm
Chapter 4
C4SD
Chapter 5
Service discovery for sensor networks
Chapter 5
DMAC
G-DMAC

Figure 1.2: Organization of the thesis. Arrows indicate specialization, while straight lines connect two related contributions.

more energy efficient, while the accelerometer-based solution is more accurate and more robust to sensor alignment problems. This work appears in the paper [1]. A demonstration shows how the sensor nodes recognize online the joint movement, using wirelessly controlled toy cars. This demonstration is described in the paper [11].

Contribution 5 ***A context-aware method for spontaneous clustering of dynamic wireless sensor nodes - Chapter 7***

We propose a method through which wireless sensor nodes organize spontaneously and transparently into clusters based on a common context, such as movement information. This algorithm is a particular case of the generalization proposed in Contribution 2. We approximate the behaviour of the algorithm using a Markov chain model and we analyse theoretically the cluster stability. We compare the theoretical approximation with simulations, by making use of experimental results reported from various field tests, including the experiments from Contribution 4. We show the tradeoff between the time history necessary to achieve a certain stability and the responsiveness of the clustering algorithm. This work appears in the paper [6].

Figure 1.2 shows the organization of the thesis. We highlight the main research directions, the contributions and the relationship among different thesis chapters. The generalized clustering algorithm proposed in Chapter 4 has four special cases, out of which C4SD and Tandem represent our contributions, described in Chapters 5 and 7, while DMAC [43] and G-DMAC [42] are shown for comparison. C4SD is used as a structural basis for a service discovery protocol designed for WSNs, while Tandem complements our proposed algorithm for the recognition of joint movement, with the final goal of clustering based on semantic properties.

Chapter 2

A classification of service discovery protocols

As described in Chapter 1, service discovery is one of the key mechanisms that allows the user to access the functionality of a dynamic and heterogeneous WSN. To be able to determine whether existing discovery solutions are applicable to the WSN environment, a survey and classification of service discovery protocols will be given in this chapter. We start with preliminary definitions and explanations of the service discovery objectives. We continue with the classification categories and sub-categories, giving definitions and examples wherever necessary. We then briefly describe the algorithms and frame them in a comparative table according to the proposed classification. Finally, we draw the conclusions.

2.1 Preliminaries

The service discovery paradigm arises in the context of self-organization in information systems, where devices featuring communication and computational resources are able to configure themselves automatically and be discovered without manual intervention. In what follows, we give the definition of the service and service discovery notions, present the service discovery entities and pinpoint the objectives of a service discovery protocol.

2.1.1 Service discovery definition

A *service* is defined as the behaviour of a system as it is perceived by its user [35]. Therefore, a service is directly related to its user, such that usability is its primary characteristic. Services may range from traditional printing, faxing or displaying images, to WSN specific, such as measuring and monitoring the environmental conditions or positioning (localization). *Service discovery* is the action of finding and locating a service in the network [110]. Given a description of a requested service, the result of service discovery is the address of one or more service providers that are able to offer the specified service. When the address is retrieved, the user may further access and use the service offered by the provider.

The environment where service discovery is performed may be composed of a variety of devices, ranging from full-fledged PCs to resource-constrained sensor nodes. Changes of service availability may happen frequently, and therefore a service discovery protocol has to provide self-configuring capabilities to accommodate these changes. Due to the adaptability and self-organization which distinguishes service discovery protocols from traditional first and second generation naming systems (such as DNS name services [129] and LDAP directory services [90], respectively), service discovery can be referred to as *third generation name discovery* [151].

2.1.2 Service discovery entities

A Service Discovery Protocol (SDP) consists of the following two participating entities:

- *The Client (or user, service consumer)*: the entity that is interested in finding and using a service.
- *The Server (or service provider)*: the entity that offers the service.

In order to facilitate discovery, it is common to find a third participating entity within SDPs:

- *Directory (or registry, server, broker, central, resolver)*: a node in the network that hosts partially or entirely the service description information in a local database, which is called *service directory* (or *service repository, service registry*).

These three entities cooperatively participate in achieving the service discovery objectives, which are described in the following sections.

2.1.3 Service discovery primary objective

Following the service discovery definition from Section 2.1.1, we identify *discovery* as the primary objective of an SDP. Discovery is the ability to find a service provider for a requested service. To achieve discovery, protocols implement the following functions:

- *Use a description language.* Services are semantically described using a certain description language. The language is used by the service provider to describe the characteristics of its services (full service descriptions), and by the service consumer for specifying the features of the requested service (possibly only a partial description). A *matching* mechanism identifies the correspondence between the requested and the provided service.

- *Store the service descriptions.* The service descriptions for each available service in the network have to be stored at particular locations (service providers and/or directories), in order to be retrieved whenever there is a request from a user.

- *Search for services.* Given a description of a requested service, an SDP has to find out the location of a service provider, by searching for the directory nodes that store a matching service description, or by directly searching for service providers.

- *Maintain up-to-date service registries.* The network must organize and deliver information about its content without human intervention. This objective translates into the following functionalities:
  - *Maintenance against changes in service description.* When services change their characteristics, an update of the service information in repositories is necessary.
  - *Maintenance against changes in service availability.* Services may become unavailable or new services may be added to the network. The result of service discovery has to change accordingly.

A functional SDP meets all of the above mentioned objectives. However, an SDP specification may be independent of the description language, covering only the last three objectives. Fusing such an SDP with an existent description language (including a matching mechanism) leads to the full specification, that can be directly implemented within the network.

2.1.4 Service discovery secondary objectives

Depending on the characteristics of each protocol, SDPs may have additional objectives, such as:

- *Functional objectives.*
  - *Service selection.* Automatic selection from a set of discovered services may be required, based on a set of metrics that is used to define the best service offer.
  - *Service usage.* Apart from performing service discovery, an SDP may also offer methods for using the discovered services.
- *Performance objectives.*
  - *Network scalability.* An SDP designed to manage large networks has to assure scalability performance.
  - *Resource-awareness.* This issue concerns designing a lightweight protocol that can be run on PDAs, mobile phones, home appliances or resource-constrained devices such as sensor nodes.
  - *Mobility support.* This feature applies to highly dynamic environments, in which nodes arbitrary may join, leave or change their position within the network. The information regarding available services needs to adapt rapidly to these changes.
- *Dependability and security objectives*
  - *Fault tolerance.* An SDP may be designed to cope with the failure of servers, being able to run backup algorithms.
  - *Security.* Blocking un-authorized access to service information can be an important factor for assuring the safe operation of an SDP.

The functional objectives mentioned above are not required for the discovery process, but they enrich the usability of SDPs. The performance objectives become important in a challenging networking environment, such as large or mobile networks, or networks composed of resource-constraint nodes. Dependability and security objectives increase the usability of SDPs in harsh and unsafe environments.

In the following, we describe the categories of our classification, taking into account the above mentioned objectives.

2.2 Classification

Firstly, we group the service discovery protocols by the network category they are designed for, as this has the most significant impact on the SDP design. Secondly, we address the objectives defined in Sections 2.1.3 and 2.1.4, by classifying SDPs depending on the type of storage and search, the description language, the service maintenance, the functional, performance, dependability and security objectives. We examine how particular objectives are met and we point out the various implementation methods.

2.2.1 Network type

The characteristics of the target network type influence the design decisions, as an SDP is required to achieve a certain performance level. We can identify the following relevant network features:

- *Size.* Network size may vary from small (i.e. one-hop wireless ad-hoc), via medium (enterprise networks) to large (wide area networks) size.
- *Throughput.* The throughput may vary from tens of kilobits per second up to terabits per second.
- *Dynamics.* Networks can be static, such as the traditional wired local area networks, or dynamic, such as wireless mobile ad-hoc networks.
- *Type of devices.* Devices participating in the network can vary from powerful servers to resource-constraint devices, such as sensor nodes.

The type of network influences the storage types chosen in the design phase of each SDP. For example, complex overlay networks are constructed for an efficient lookup in large and relatively stable networks [86]. Centralized solutions are suitable for small networks composed of powerful devices [127]. Mobile ad-hoc networks may choose unstructured distributed storage, in order to minimize the traffic generated by mobility [74]. More information can be found in Section 2.2.2, which presents a detailed view of the storage structures used by SDPs.

2.2.2 Storage of service information

Information retrieval of available services relies on the storage system type. We argue that given the network type, storage is one of the most important classification criteria, as it directly influences the performance of SDPs in terms of scalability, mobility support and resource awareness. Depending on the network type, various storage systems can be designed. For example, in ad-hoc networks, storage may be inexistent due to increased mobility, whereas in wide area networks it is compulsory to have intermediate storage, although this can increase substantially the design complexity. We identify the following major storage types:

- *Centralized.* This approach is optimal for rapid access to data and low traffic, even though it creates a single point of failure. SDPs usually use the server only for information retrieval, the actual communication between the client and the server being done in a peer-to-peer manner [127].

- *Unstructured Distributed.* In unstructured distributed storage systems, communication is based on broadcast or multicast mechanisms. This technique is common for protocols designed to work in local area networks and ad-hoc networks. Typically, every node has a local service directory maintained as a limited-time cache. To obtain the service data, service providers flood the network with service advertisements and clients broadcast discovery messages. The cached service information comes from service advertisements and replies to discovery messages. The clients and intermediary nodes use the local service database for generating replies [158].

- *Structured Distributed.* Structured distributed storage methods are commonly found in the context of large networks, where the storage solutions mentioned earlier do not scale well. Directory nodes organize themselves in an overlay structure that allows them to route the discovery messages in a limited number of hops. We classify the structured distributed systems into three categories: hierarchical, flat and hybrid.

  – *Hierarchical.* This type of storage follows the DNS [129] model. Information, advertisements and queries are propagated up and down through the hierarchy. Parents store information of their children and the search flow is directed to the root node. Therefore, the root node can become a bottleneck. If the size of network is large, a compression method for the stored information is necessary [86].

  – *Flat.* Protocols that fall in this category rely heavily on peer-to-peer overlay networks, constructed by means of *distributed hash tables (DHT)*, such as CAN [138], Chord [147], Pastry [141] and Tapestry [169]. DHTs are used to store key-value pairs on designated nodes.

Messages are input to a hash function and routed in a bounded number of hops to the nodes responsible for the resulting key. Each node maintains a routing table with identifiers and network addresses of other nodes. The major advantage of DHT protocols is the efficient lookup mechanism, which normally is performed within $O(log(N))$ hops, where N is the number of nodes in the overlay network.

– *Hybrid.* Hybrid solutions combine ideas from the above hierarchical and flat storage mechanisms with additional optimization techniques. Some of them rely on hierarchical ring models to organize groups of nodes [100]. Others try to overcome the disadvantages of the DHT approaches (e.g. the cost of maintaining a consistent distributed index), while preserving their benefits (e.g. the efficient lookup mechanism) [155]. Clustering algorithms provide a local hierarchical model, combined with the global unstructured peer-to-peer [4] or spanning-tree model [102]

Centralized storage solutions are typically found in small to medium size local area networks, where service registries are usually available. Unstructured distributed storage is commonly used within infrastructure-less environments, such as ad-hoc networks, because it distributes the registrations among all the nodes and it requires minimum overhead. However, unstructured distributed storage may lead to a high discovery cost. This inconvenient is addressed by structured distributed storage, where the efficiency of service lookup comes at the cost of maintenance overhead. Hybrid solutions try to balance the maintenance and discovery costs by merging various structured and unstructured techniques.

2.2.3 Search methods

Depending on what type of storage each protocol chooses, different search mechanisms can be identified. The object of discovery can be:

- *Directory node.* In the centralized and structured distributed storage environments, clients and servers need to discover the directory nodes for sending their advertisements and requests.

- *Directory nodes in the overlay structure.* In the structured distributed storage systems, directory nodes need to route service discovery messages to other directory nodes in the overlay structure.

- *Services.* In unstructured distributed storage systems, nodes have to find the appropriate services without the help of directory nodes.

It is important to mention that the extension of the search is conditioned by the dispersion degree of the information in the network. The threshold between the initial dissemination of service descriptions and the extension of the following queries needs to be taken into account. On the one hand, more organized and distributed information translates into less search effort. On the other hand, complex storage mechanisms make the information consistency difficult to maintain. That is why, in highly mobile networks, flooding may be the only option for service lookup.

The main two types of search methods are:

- *Passive Discovery (or Push Model).* A server announces the services that it offers by sending advertisement messages to potential clients. A directory announces its presence, so that servers can register their services.
- *Active Discovery (or Pull Model).* A client that needs information about services or brokers sends discovery messages. A server sends discovery messages to locate the potential directory nodes.

In general, protocols implement both methods. Advertised service descriptions can be the local ones [73], or the entire local database, including services offered by others [131].

Depending on the directory structure, we have the following search-flow types:

- *Flood-based.* This search type is present in unstructured distributed storage. To obtain the service data, a service provider sends service advertisements and a client sends discovery messages. The flooding is either limited (e.g. to a number of hops) or it covers the whole network.
- *Directed flow.* Complex search is effectuated mainly within structured distributed SDPs. Search queries flow based on the rules specific to each protocol. For example, in hierarchical approaches, data flow up and down the hierarchy [86].
- *Hybrid.* Hybrid SDPs use both flooding and directed flow. For instance, SDPs relying on clustering use directed flow for intra-cluster communication and flood-based search for inter-cluster discovery [4].

Flood-based search is typically common with SDPs that employ unstructured distributed storage, whereas directed flow is used with structured distributed storage. Hybrid distributed storage is usually associated with a hybrid search flow.

2.2.4 Service description

Service discovery protocols can be independent of any particular type of service description, or they can provide the complete solution, including the specifications of a description language. We identify the following description alternatives:

- *Textual.* A service can be described using a textual description. An SDP chooses a set of keywords and associates them with key values, such that the search engines will search for keys using a set of query keywords [53].

- *Attribute-value pairs.* The most widely used description format is the attribute-value structure. An attribute is a category in which an service can be classified, for example the *resolution* of a photography service. A value is the classification within that category, for example, 640×480 [29].

- *Hierarchy of attribute-value.* Some protocols use a hierarchical arrangement of attribute-value pairs, such that an attribute-value pair that is dependent on another is a descendant of it [29].

- *Markup languages.* SDPs [34, 55] may use XML schemas for having valid attribute definitions, such RDF [21] or DAML+OIL [16].

- *Object-oriented interface.* A service can be described using an object-oriented programming interface. For example, Jini requires that service descriptions are expressed in the form of Java interfaces [127].

On the one hand, complex description languages allow for a detailed characterization of services and thus facilitate the service selection (see Section 2.2.6). On the other hand, communicating, storing and processing thorough descriptions increase resource utilization and may be infeasible for resource-constraint environments, such as wireless sensor networks. However, if detailed descriptions are required by applications, compression techniques can be used to minimize resource utilization [7].

2.2.5 Service maintenance

Service maintenance is regarded as the permanent adjustment of the service information stored on directory nodes. We present the existing solutions for the two types of maintenance mentioned in Section 2.1:

- *Maintenance against changes in service description.* The storage system maintains consistency against changes in service characteristics by using the following methods:
  - *Service advertisements.* A node adjusts the service information according to newly advertised descriptions. This technique is mostly used in unstructured distributed systems, where all the nodes receive the service advertisements, independently of their interest in the service.
  - *Event notification or publish/subscribe.* A server publishes its offer and interested clients subscribe for events with directory nodes or directly with the server. Service updates are received only by the nodes that have subscribed for the service.
- *Maintenance against changes in service availability.* Services can be added or deleted from the network or network topology may change, while directories have to preserve a consistent view of the available services. The following schemes are used to control service availability:
  - *Passive maintenance.* Passive maintenance assigns the responsibility for maintaining service registrations to the server. We have the following types of passive maintenance:
    - *Soft state.* Servers have to periodically re-register their services in order to restate the service availability; if within a certain amount of time no re-registration is received, directories delete the old service registrations.
    - *Hard state.* Service registrations do not expire in a specific amount of time; they remain unchanged until they are explicitly deleted. Deletion may occur when servers leave the network and send explicit de-registration messages.
    - *Hybrid state.* Some protocols may implement hybrid state management for combining the management-simplicity of soft state to the low-bandwidth requirements of hard-state. For example,

directory nodes at the edge of the network refresh state information at a higher frequency than those part of the network core [39].

- *Active maintenance: polling.* Active maintenance gives the responsibility of maintaining a consistent service registry to the directory nodes. A common technique is polling, where directory nodes periodically check service availability.

Soft state techniques and polling induce a high maintenance cost in terms of traffic, inconvenient which is avoided by hard state methods. However, applying hard state maintenance may lead to delays in achieving consistency of service registries. Hard state maintenance that uses networking information for a rapid identification of server unavailability or hybrid state methods are two options for minimizing maintenance overhead and achieving fast convergence of service registries.

2.2.6 Service selection

After submitting a query for a certain service, it is often the case that multiple servers can offer the specified service. The best service can be chosen in the following ways:

- *Manually.* The user manually selects the server out of a list of service providers. This is the most used method, as it does not require any protocol implementation.
- *Selected by the client.* An optimization algorithm implemented on the client's side can automatically choose the best server.
- *Selected by the directory.* The best server can be selected by one of the directory nodes present in the system.

When considering the last two cases, an important issue is the metric used to define the best offer. Metrics generally depend on service performance parameters or context attributes, such as lowest hop count, smallest response time, least loaded node, best channel conditions etc.

2.2.7 Service usage

Although the main goal of an SDP is to provide the address of the server that offers a particular service, some SDPs may offer also a mechanism for service usage. Commands can be given using RPC, Java RMI, Authenticated RMI, SOAP etc. For example, an SDP based on HTTP for communication and on XML for descriptions may use SOAP for service usage [73]. Java RMI can be used for communication between services, which allows not only data to be passed from object to object around the network but also full objects, including code [127]. Other protocols utilize generic *code binding* to associate a service description with the implementation code [53].

2.2.8 Network scalability

Scalability is one issue tightly related to load balancing and query efficiency. We have the following types of scalability problems:

- *Storage.* Directories may be burdened with significant storage of service information.
- *Traffic.* Directories and servers may be overwhelmed by registration messages and queries.
- *Delay.* Searching in large networks may induce high delay in the retrieval of the desired information.

SDPs try to avoid the circumstances that lead to the above-mentioned scalability problems. We present in the following the methods that have been proposed for achieving scalability:

- *Service grouping - storage and traffic.* Protocols may address the scalability issue by enabling service grouping and limiting the queries to groups of nodes [81].
- *Caching - traffic and delay.* This technique prevents the directory nodes responsible for storing information associated to a service from being overloaded if the popularity of the service increases [53]. Caching also reduces the delay, since a node closer to the user may retrieve the address of the service provider from its local cache.
- *Load distribution - storage and traffic.* If a directory node is overwhelmed by service registration and queries, it may delegate other nodes to join the distributed directory and take over a part of its effort [86].

- *Hierarchical structures - traffic and delay.* DNS-like hierarchical structures enable network scaling to a large number of nodes. Parents store information about their children and queries pass up through the hierarchy until a possible match is encountered. After that, requests are routed down to the child offering the specified service [86]. When a node is overwhelmed by the amount of information it is supposed to store or is flooded with queries, it can easily spawn the load on a nearby machine, assigned to be its child. On the contrary, if a node considers that it is insufficiently burdened, it can delegate the load to its parent.

- *Distributed hash tables (DHT) - traffic and delay.* DHT approaches provide an efficient and scalable index lookup mechanism. Messages are routed in $O(logN)$ hops, where N is the number of resolvers in the network [140].

- *Overlay structure optimization - traffic and delay.* To improve efficiency in overlay networks, some protocols adapt to network changes by optimizing the network structure [99]. The overlay is adjusted to provide the optimal number and location of directory nodes, for a fair sharing of query loads and minimum delay.

- *Aggregation of descriptions - storage and traffic.* For reducing the storage of service descriptions, an SDP may utilize compression methods [86]. Aggregation of service advertisements and registrations can also be used to reduce the network traffic: instead of sending a message for each advertisement, a collection of descriptions can be advertised in one single packet [108].

- *Clustering - traffic.* Grouping nodes into clusters leads to restricted communication and data exchange, which improves on network scalability [4].

SDPs may implement more than one of the above techniques to improve scalability. For example, load distribution comes in line with DHT structures, where an overloaded node invites other nodes to become part of the directory structure, and delegates part of its tasks to these nodes [39]. Hierarchical structures facilitate aggregation of descriptions, where summaries of service descriptions are registered at superior hierarchical levels [86].

2.2.9 Resource awareness

Only few SDPs are designed to take the available node resources into account. Protocols that support integration of resource-constraint devices usually follow one of the strategies mentioned below:

- *Workload delegation.* Only a part of the network nodes are capable of running the SDP. The nodes that cannot handle the load, delegate part of their tasks to the more powerful nodes in the network [152].
- *Workload distribution.* All the network nodes are capable of running the SDP, but the workload is distributed among the nodes based on their capabilities [4].

Delegation of work load is useful when there are powerful nodes that can handle increased overhead. However, this technique may not be a solution when the powerful devices are scarce and hard to reach. In this case, the solution is to implement a lightweight SDP that is able to run on resource-constraint devices, and that distributes the workload (typically service registrations and queries) depending on node capabilities.

2.2.10 Mobility support

Weak mobility support is implemented by most protocols along with maintenance (see Section 2.2.5). However, in networks where dynamics is the primary characteristic, regular maintenance techniques are not sufficient for keeping the information up-to-date.

The two basic strategies for solving this problem are the following:

1. *Reactive.* The information changes according to the events in the network (e.g. the route to server is no longer available).
2. *Proactive.* Nodes maintain a consistent view by periodically exchanging update messages.

Using these strategies and depending on the storage type that an SDP is maintaining, the following methods have been proposed for handling mobility:

- *Adjustment of service advertisement rate and zone radius.* A small advertisement time interval can be implemented for highly dynamic environments, opposed to a larger value for rather stable networks. The zone where proactive information is maintained, or the radius of service advertisements (e.g. the number of hops) can be regulated depending on different network dynamics [55, 85].

- *Aggregation and filtering.* For preventing directories to keep refreshing the status of services when they move around, aggregation and filtering is used at service registration and service discovery phases. A service will register only if there is no other server within the database that offers the same service [170].
- *Using routing information.* The routing information is useful to determine whether certain nodes and their services were added or disappeared to/from the network [158].
- *Late binding.* The late binding technique integrates the discovery messages with routing, such that the address of the service is not returned to the client, but instead the directory nodes forward the message to the server. In this way, seamless communication between clients and services is enabled, even if the location of the service changes during service usage [29].
- *Overlay network restructuring.* Structured distributed systems that build overlay networks have to follow certain structural conditions and therefore implement additional algorithms for preserving consistency in the case of nodes entering or leaving the network, and in the situation of network partition and reintegration [100].
- *Clustering.* The clustering technique can be used to partition the network with the objective of maintaining a relatively stable topology, by choosing the less mobile nodes as clusterheads. The clusterheads act as directories for the nodes in their cluster [4].

Mobility is a factor that greatly influences the maintenance overhead. The more mobile the network is, the more traffic is required to keep the consistency of service registries. For this reason, loose directory structures (unstructured distributed or hybrid storage) are preferred in a highly dynamic environment.

2.2.11 Fault tolerance

Fault tolerance is defined as the mechanism to avoid system failures in the presence of faults [35]. In this section, a fault is referred to as the failure of directory nodes. Since SDPs are designed as self-managing systems, failure of directory nodes should be handled for inconspicuous system recovery.

We describe a number of solutions for fault handling that have been designed for SDPs, following the taxonomy of Avižienis et al. [35]:

- *Compensation.* Compensation is an error handling mechanism where the erroneous state contains enough redundancy to enable the error to be masked. For example, multiple copies of the service information are maintained on several resolver nodes. When one of the resolver nodes fails, the remaining directories can still respond to the query [29].

- *Isolation.* Isolation performs physical or logical exclusion of the faulty components from further participation in service discovery. The faulty directory nodes will be identified and removed from the SDP operation [152].

- *Reconfiguration.* Reconfiguration either switches in spare components or reassigns tasks among non-failed components. For example, in centralized storage solution, a backup of the central directory node is maintained, such that when the central fails, the backup takes over the brokering function [152]. For distributed storage, new nodes will dynamically take over the directory function from the faulty resolver [4]. Reconfiguration occurs also in case of multi-modal functioning of SDPs: the protocol has an optional peer-to-peer operation, which is enabled in case of directory failure [81].

- *Reinitialization.* Reinintialization checks, updates and records the new configuration and updates system tables and records. Within SDPs, reinitialization represents the re-election of the directory node and re-registration of services [152].

With distributed SDPs, the redundancy of service registries facilitates fault tolerance through compensation and reconfiguration. Centralized SDPs need explicit backup mechanisms in order to be able to mask and recover from errors [152].

2.2.12 Security

The primary objective of service discovery is to find service providers for a desired service (see Section 2.1.3). Any intended action that can be performed by an entity and that can prevent the achievement of this goal is considered a security threat. Following the taxonomy of Avižienis et al. [35], we present in the following the security attributes that represent the objectives of any secure system. In particular, we show how these attributes are addressed by secure SDPs.

Primary attributes

- *Availability.* Availability is the property of a system or a system resource being accessible and usable upon demand by an authorized system entity, according to performance specifications for the system [143]. For example, an attack to availability is the *denial of service*, where malicious nodes can flood the network with false discovery messages. The resources of legitimate nodes may be depleted in an attempt to process such messages, and thus the entire service discovery process could be damaged [110].
- *Confidentiality.* Confidentiality is the property that information is not made available or disclosed to an unauthorized entity [143]. For example, a malicious node can intercept or eavesdrop on service discovery messages and obtain the identities of the senders and intended recipients and the specific services that are being requested or advertised [110]. To prevent the information disclosure, SDP messages can be encrypted using, for example, public key and symmetric key encryption [86].
- *Integrity.* Integrity is the property that data has not been changed, destroyed, or lost in an unauthorized or accidental manner [143]. More specifically, the messages exchanged during service discovery should not be modified. SDP message integrity can be achieved through verification mechanisms, such as hashes [170].

Besides the primary security attributes mentioned above, secondary attributes can be defined, which refine or specialize the primary attributes [35, 51]. In the following, we give a list of secondary attributes that are addressed by SDPs:

Secondary attributes

- *Authenticity.* Authenticity is the property of being genuine and able to be verified and be trusted [143]. For a message, authenticity is equivalent to integrity of both the message content and of the message origin. A user is authentic if the declared identity is the real identity of that person.

Authentication is the process that gives confidence in authenticity [51]. SDPs may implement mutual authentication among servers, clients and directories. Clients may have to authenticate in order to use a service [61] or directories have to authenticate to clients [81]. The full procedure is that all endpoints have to authenticate to each other [86]. Options for achieving authentication include username and password methods [61], digital signatures or challenge-response protocols.

- *Authorization.* Authorization is defined as the right or permission that is granted to an entity to access a system state [143]. In the context of service discovery, authorization establishes the rights of the service discovery entities to access service information. Options for achieving authorization include the usage of capabilities [86], access control lists [127] and credentials [155].

- *Non-repudiability.* Non-repudiability is a security attribute that represents the availability and integrity of the identity of the sender of a message (non-repudidation of the origin), or of the receiver (non-repudiation of the reception) [35]. *Non-repudiation* provides protection against false denial of involvement in a communication [143]. It can be achieved for example by hashing the data and signing the hashes before encryption [170].

- *Privacy.* Privacy is confidentiality with respect to personal data [51]. In the context of service discovery, privacy assures that service information is not divulged to external entities that are not participating in the service discovery process [110]. For instance, the location information of users is kept private until they want to reveal their position [170].

Security is of primary importance when SDPs function in public environments. SDPs designed for these environments need to implement a full-fledged security strategy, which includes most of the above mentioned attributes [170].

2.3 Algorithm description

In this section, we briefly describe the SDPs based on the different strategies of storing the service information, presented in Section 2.2.2. The storage type directly influences the search method and the performance of an SDP.

2.3.1 Centralized

Jini [127] is based on the Java RMI technologies and Java computing platform. Services are defined by Java proxy objects and are registered in a central location called lookup service. When a service registers its proxy object, it is returned a lease. The service must renew the lease after a certain amount of time to sustain its presence in the lookup service. Services can be grouped into logical sets governed by certain lookup services in the network. When a client needs a service, it first contacts a lookup service and asks to find one or more services that match a template. The lookup service will return an appropriate proxy object. The client will then be able to use the service by method invocation. Small devices are able to run Surrogate Jini [128]. They use host-capable machines that have the computational resources to execute code written in Java on their behalf.

Splendor [170] proposes an SDP particularly designed for public environments. Splendor defines the following entities: clients, services, directories and proxies. Proxies offer security and privacy services. Other services ask proxies to handle registration, authentication, authorization and key management for them. Directories register service descriptions and answer the queries of clients. Directories cache soft state information of mobile services and hard-state information about proxies. Splendor addresses the mobility problem by using aggregation and filtering at the service registration and sevice discovery phases.

SLP [81] allows for two modalities of operation: centralized and unstructured distributed. SLP uses the terms *User Agent* (UA) for the client, *Service Agent* (SA) for the server and *Directory Agent* (DA) for directory entity. When a DA is present, it collects all service information advertised by SAs, and UAs unicast their requests to the DA. In absence of a DA, UAs repeatedly multicast the same request they would have unicast to a DA. SAs listen for these multicaset requests and unicast responses to the UA if they have the requested service. Service templates (documents registered with Internet Assigned Numbers Authority (IANA) [18]) define the attributes of services, their default values and interpretation. To handle large network environments, more DAs can share the service registrations, or logical groups of services can be defined, using *scopes*. Discovery and lookup can be performed within the scope administrated by a designated DA.

Salutation [61] defines an entity called *Salutation Manager* (SLM), which functions as a service broker. A device in the network is called *Networked Entity*, which can be either a client or a service. The SLM contains a registry to hold information about services. The SLM can discover other remote SLMs and determine the services registered there. The SLM can periodically check the availability of a service. Also, clients can request event notifications regarding services, called *long-term requests*. The description of a service is an *Attribute Record*, which consists of an attribute ID, a compare function ID and a value field.

FRODO [152] is an SDP designed for home environments. FRODO defines three types of device classes: (1) *3C* are simple devices that implement only the network stack to connect to the system, (2)*3D* are medium devices that implement the network stack and SDP, and (3) *300D* are powerful devices. The storage system is a hybrid between centralized and unstructured distributed. If there is a 300D device in the network, then this device will take the role as *Central*, or directory node. Otherwise, the SDP will function in a distributed way. A leader election algorithm elects the most powerful device as Central, while the next device on the list is appointed as *Backup*, which will take over in case of Central failure.

2.3.2 Unstructured distributed

UPnP [73] is based on a completely distributed peer-to-peer protocol. The foundation of UPnP networking is IP addressing, such that devices get their IP addresses via DHCP or AutoIP. Devices advertise their services by multicasting advertisement messages to a standard address and port. Clients (or control points) can also search for devices and services of interest in the network, by multicasting discovery messages. Eventing in UPnP consists of services publishing updates and control points subscribing to receive this information. The UPnP descriptions are in XML syntax and are following the standard UPnP Device Template.

Bluetooth SDP [60] defines a client-server interaction between devices. Every Bluetooth device may function both as a client and as a server, depending on whether the device is both a service consumer and a service provider. The server maintains a list of service records that describe the characteristics of services associated with the server. A client may retrieve information from a service record maintained by the SDP server by issuing an SDP request. A service record consists of a list of service attributes.

DEAPspace [131] is an SDP designed for wireless ad-hoc single-hop networks. Each device maintains a view of all services that are available in the network. The nodes advertise not only their own services, but they send a full list of all the known services. Therefore, occasionally missing packets do not cause significant problems, and also the system converges rapidly to consistence.

Konark [84] is a completely distributed SDP designed for ad-hoc, peer-to-peer networks. Each device has a local repository where it maintains the local services. For sharing this information with other devices, servers send service advertisements to the network, which contain the time-to-live information for keeping the system update. Clients cache these advertisements for further usage. Services are described using XML and the user can interact with services by invoking the available functions via SOAP.

Other protocols that implement peer to peer caching are Allia [139] and GSD [55]. Allia [139] defines an *Alliance* of a particular node as the set of nodes whose local service information is cached by this node. When a node needs to discover a certain service, it first looks at its local cache to check the members of his alliance. If the service is unavailable, the node sends the request to other alliances in its vicinity. The advertisement frequency and the alliance diameter depend on the mobility of nodes.

In GSD [55], each server periodically advertises a list of its services to all the nodes in a particular number of hops, called the diameter of advertisements. The advertisement frequency, diameter and lifetime are user-controlled parameters, that can be specified according to the network mobility level. The services are grouped depending on their functionality. The service group information is propagated along with the service advertisements. This information is used to selectively forward the service discovery messages to the nodes that are part of a specific group.

Varshavsky et al. [158] propose a cross-layer discovery protocol that uses routing information for making decisions of service availability. The routing protocols that are considered are DSR [93] and DSDV [136]. The design consists of two main components: the routing-independent Service Discovery Library (SDL) and the Routing Layer Driver (RLD). SDL stores information about known servers in a service table. Clients and servers call on SDL to issue service discovery requests and propagate service advertisements. SDL calls on RLD to disseminate service discovery requests and advertisements, and propagate service discovery replies. RLD forwards service discovery messages it intercepts from the network to SDL and informs it about changes in network topology.

Using the same cross-layer idea as Varshavsky et al., Frank and Karl [74] implement the service discovery functionality on top of AODV [137] routing protocol. When a client issues a request, a routing packet including the description of the service is created. The message propagates through the network like a normal AODV routing packet. A node that receives it and knows a matching service provider, fills in the destination address. Nodes know about service providers via service advertisements and caching. Negative service announcements are used to explicitly remove cache entries and thus achieve a faster consistency convergence.

The resolver network in INS [29] is comprised of *Intentional Name Resolvers (INRs)* organized in a spanning tree, which is only used for disseminating service descriptions (i.e. is not used for organizing a structured directory). The service descriptions are called *name-specifiers* and consist of a hierarchical arrangement of attribute-value pairs. INS uses the late binding technique for handling mobility, by integrating the discovery messages with routing. Services periodically advertise their intentional names to the INR network. INRs replicate the name information among each other using a routing protocol that includes periodic and triggered updates. To handle excessive lookup loads, INRs spawn instances on other candidate resolvers. There is a central component in the network, named Domain Space Resolver (DSR), which maintains a list of active and candidate INRs. For scalability reasons, services are grouped into *virtual spaces*, such that every INR only needs to route a subset of all the active virtual spaces in the system.

Lenders et al. [108] propose a service discovery protocol inspired by electrostatic fields from physics. Nodes in the ad-hoc network determine the *potential* of a service depending on the distance to service providers. Service instances periodically advertise the service descriptions they offer. These advertisements are flooded through the network within a limited scope. Each node stores the advertisements in a time-limited cache and calculates the potential value for each service type. A service request packet arriving at a node is forwarded to the neighbour with the highest potential. For reducing the communication overhead, nodes aggregate more advertisements and forward them in one single message.

2.3.3 Structured distributed

Following the classification from Section 2.2.2, a structured distributed storage can be either flat, hierarchical or hybrid. We describe different methods for structured distributed storage by giving examples of SDPs for each category in turn.

2.3.3.1 Flat

In INS/Twine [39] achieves scalability via hash-based partitioning of resource descriptions among a set of directory nodes, or *peer resolvers*. The service descriptions are hierarchies of attribute-value pairs. INS/Twine splits service descriptions into *strands* and computes hash values from each strand. The resulting set of numeric keys are then stored in the network of resolvers. INS/Twine uses Chord [147] for the underlying DHT process. To maintain consistency in face of changes, INS/Twine uses a hybrid state policy, where resolvers at the edge of the network refresh state information at a higher frequency than resolvers in the network core.

The Superstring service discovery protocol [140] also uses a DHT data structure [147]. A service sends its description to the nearest resolver, which produces a key from the top-level component of the description. The key is used to route the description to the appropriate resolver. In turn, this resolver removes the original top-level component and hashes each of the top-level components of the resulting sub-trees. This resolver then sends the descriptions of the sub-trees further to other resolvers, depending on the hashes it produces. The process is repeated until the bottom of the description is reached. Thus, the computational cost for any query is shared among many resolvers.

CDS [75] constructs a DHT overlay network, out of which a small number of nodes, the *Rendezvous Points (RP)*, are in charge of registering the service descriptions. A hash function is applied to each attribute-value pair in the description and the resulting values are registered with the Rendezvous Points. The problem is that the attribute-value pairs may overlap for different service descriptions, such that the query load is concentrated to one node. This problem is addressed by constructing a load balancing matrix of RPs, which grows and shrinks dynamically, as a result of the registration and query load. Each column contains a partition of the content and nodes in the same column are replicas of each other. CDS allows the client to make use of a query optimization algorithm, which selects the best RP. Two criteria can be taken into account: the RP with the smallest response time or the RP with the smallest database.

One Ring Rules Them All [53] uses structured peer-to-peer overlays as a platform for service discovery and chooses Pastry [141] as an example for distributed search. The infrastructure proposed relies on a universal ring that all participating nodes are expected to join. The ring stores the keys that provide access to the information about services, the code needed to run them and contact lists of server nodes. The storage system is persistent, except the contact lists. Caching is used to avoid overloaded nodes.

2.3.3.2 Hierarchical

SSDS [86] is a protocol designed for wide-area networks. It achieves scalability by constructing a global hierarchical structure. Summaries of service descriptions are registered at the superior hierarchical levels using Bloom-filtered cross terminals [47]. The result of a Bloom filter is a bit vector that summarizes aggregated descriptions. If a query comes up the hierarchy, the receiving directory checks to see if it hits locally or in any of its children. If so, the query is routed down to the children. Otherwise, the query is passed upward.

CSP [107] also uses a hierarchical structure of directories and aggregation of service descriptions, based on the Centroid aggregation method. This approach exploits the word frequency pattern followed by the Zipf distribution [112]. CSP specifies that seldom used words need to be discriminated in favour of more frequently used ones. The attributes that are not very spread encounter aggressive aggregation that forces service information to quickly converge to a small size set. CSP introduces the concepts of static and dynamic context-attributes for service selection. Static attributes are the ones that keep fixed values set at the time of announcements, while the value of a dynamic attribute is determined at the time of lookups. Examples of context attributes are: distance to server, server load and channel conditions. The evaluation of context attributes is done by the directory nodes (Broker Agents).

GloServ [34] is a service discovery protocol for local-area and wide-area pervasive computing. Services are defined using Resource Description Framework (RDF) [21]. GloServ uses a hierarchical structure, where every directory in the hierarchy has its own RDF schema store which describes all the services of its children.

2.3.3.3 Hybrid

The Project JXTA [155] establishes a virtual DHT network overlay on top of existing physical network infrastructure. This approach combines DHT methods with a random walker that checks for non-synchronized indices. The resolvers *Rendezvous peers* are nodes that store service advertisements. Rendezvous peers are not required to maintain a consistent distributed hash index, in order to avoid expensive network traffic. Instead, a limited-range walker is used to walk the rendezvous from the initial DHT target.

Service Rings protocol [100] uses a hierarchical ring structure to achieve scalability. A ring is a group of devices that are physically close to each other and offer similar services. Each service ring has a designated Service Access Point (SAP), which stores information about all the services offered by the ring. SAPs can be organized in higher-level rings, which also have SAPs that store summaries of services they provide. Rings permanently monitor the network traffic and make decisions for optimizing their structure, by running algorithms for ring restructuring, splitting and merging.

LANES [99] builds up an overlay network inspired by CAN [138]. The two-dimensional CAN structure is optimized by constructing loosely coupled lanes of nodes. An arbitrary node from the lane has full information of the services offered by that lane. Service announcements are propagated throughout a lane, whereas service requests are sent to other lanes. On average, this algorithm distributes the descriptions to $\sqrt{N}$ nodes, where N is the total number of nodes in the network. LANES deals with inefficient inner lane connections and lanes splitting and merging to achieve the optimal size.

Overlay networks can be built also by making use of backbone selection algorithms. Kozat and Tassiulas [102] tackle the solution of a distributed directory by creating and maintaining a network backbone. Selected nodes are considered to form a relatively stable dominating set. They receive and process service requests, acting as service discovery agents. However, due to the high density of nodes in the dominating set, a lot of loops are generated in the discovery process. Therefore, a source-based multicast tree algorithm is proposed to organize the nodes in the backbone.

Zone-based protocols and cluster-based protocols proactively maintain routing and service information inside the zone/cluster, while using a reactive search method at the network level. Helmy [85] proposes a resource discovery protocol, where each node keeps track of a number of nodes less than R hops away, which defines the *zone* of the node. As part of the zone information each node maintains resource information and routes to all the nodes in its zone. A node has also knowledge of a number of contact nodes outside its zone. The search method implies forwarding the requests to the contact nodes, which is analogous with flooding between contacts.

SD4WSN (see Chapter 5) proposes a service discovery protocol suitable for heterogeneous wireless sensor networks, which reduces the workload of the resource-constraint devices. SD4WSN is based on clustering, where a set of nodes, selected based on their capabilities, acts as a distributed directory of service registrations for the nodes in their cluster. In this way, the communication costs are reduced, since the service discovery messages are exchanged only among the directory nodes, and the distribution of workload takes into account the capabilities of the nodes.

2.4 Comparative table

Table 2.1 gives a comparison among the SDPs described above, following the proposed classification of Section 2.2. Similarly to the description of algorithms, we group the SDPs based on the storage type. Dashes correspond to "not applicable" features.

A close analysis of Table 2.1 reveals the following characteristics of SDPs:

- *Network type vs. storage.* Enterprise and ad-hoc networks make use of centralized and unstructured distributed SDPs, due to the relatively small network size that does not pose scalability problems. Wide-area networks use mostly flat or hierarchical structured distributed storage that meet the scalability requirement through an efficient lookup mechanism. Mobile ad-hoc networks generally use less structured storage types (unstructured distributed or hybrid structured distributed), in order to reduce the maintenance effort.

- *Search type vs. storage.* Unstructured distributed storage is typically flood-based, while structured distributed storage uses a directed flow search. Hybrid storage SDPs may use both flooding and directed flow, which represents a hybrid search (see Section 2.2.3). Active and passive search are commonly used in a joint manner.

- *Description vs. usage.* When an SDP specification includes a description language, it often incorporates a usage mechanism that is in line with the chosen description. For example, with UPnP, an XML schema defines the UPnP language, and SOAP is employed for service usage [73]. Services in Jini are described using Java interfaces, while Java RMI represents the service usage mechanism [127] (see Section 2.2.4). However, many SDPs are independent of description languages and usage mechanisms.

- *Security vs. network type.* Security is addressed mostly by SDPs designed for enterprise networks, public spaces and wide-area networks.

- *Scalability vs. network type.* Scalability is a great concern for SDPs designed for ad-hoc, sensor and wide area networks. Wide-area networks achieve scalability mainly through structured storage, while ad-hoc and sensor networks use mostly caching and clustering.

- *Fault tolerance vs. storage.* Compensation is mainly used with distributed storage, where replicas of service registries facilitate the directory failures to be masked.

By analysing Table 2.1 following the classification criteria, we observe that the *resource awareness* objective is often neglected, most SDPs being designed to function on resource-rich devices. However, discovery is one of the key mechanisms that allows the user to access the functionality of a dynamic and heterogeneous WSN, as described in Chapter 1. A WSN is typically resource-constraint, which makes existing SDPs less appropriate for this type of environment. In the following, we explain why existing SDPs are not suitable for WSNs.

As indicated earlier, SDPs which achieve efficient service lookup in large-scale ad hoc networks exploit either flat or hierarchical overlay structures. These techniques generate considerable network traffic and high maintenance overhead, so they are not suitable for running on resource-lean devices.

More appropriate for this environment are the protocols that use unstructured distributed storage [84, 55]. The service information is obtained using either the *push model*, where service providers advertise periodically the services they offer, and/or the *pull model*, where clients flood the network with discovery messages in search for the desired service. However, flooding is a method that limits the scalability of protocols, generating substantial traffic especially for dense networks.

Cluster-based protocols proactively maintain routing and service information inside the cluster, while using a reactive search method at the network level. As a consequence, they reduce the network traffic compared to flood-based protocols, which is important especially in networks with high density. However, the existing cluster-based SDPs require a high cost for maintaining extensive topological knowledge or complex structures. For example, with Helmy [85], nodes need to maintain a complete topological view over a number of hops, together with the knowledge on available resources. Kozat and Tassiulas [102] propose an SDP where the clustering structure is composed of two overlays, i.e. the dominating set and the multicast tree. This approach is expensive for resource-constraint devices.

Nevertheless, using lightweight clustering algorithms remains a promising solution to achieve resource-awareness in service discovery, because of their ability to avoid the negative effect of flooding and to achieve workload distribution. We provide detail on future research directions in the next section.

Table 2.1: Overview of service discovery protocols

SDPs	Network type	Storage	Search	Description	Maintenance	Selection and usage	Scalability	Resource awareness	Mobility and fault tolerance	Security
Jini [127]	Enterprise network	Centralized	Active and passive	Java interfaces	Event notification Soft state	Java RMI	Service grouping	Workload delegation [128]	-	Authentication Authorization Confidentiality Integrity
Splendor [170]	Public environments	Centralized	Active and passive	-	Soft-state and hard-state	-	-	-	Aggregation and filtering	Authentication Authorization Confidentiality Integrity Non-repudiation
SLP [81]	Enterprise network	Centralized and unstructured distributed	Active and passive	Attribute-value (IANA)	Soft state	-	More directories Service grouping	-	Reconfiguration	Optional authentication
Salutation [61]	Enterprise network	Centralized or unstructured distributed	Active	Attribute-value	Event notification, polling	RPC	-	-	-	User authentication
FRODO [152]	Home appliances	Centralized and unstructured distributed	Passive	-	Soft state and polling Event notification	Best match	-	Workload delegation	Isolation Reconfiguration Reinitialization	-
UPnP [73]	Enterprise network	Unstructured distributed	Active and passive Flood-based	XML (UPnP language)	Service adv. Event notification Soft state	SOAP	-	-	-	Integrity Authentication Authorization
Bluetooth SDP [60]	Small, max. 8 devices	Unstructured distributed	Active	Attribute-value	-	-	-	-	-	Optional authentication, authorization, partial confidentiality
DEAP space [131]	Ad-hoc single-hop	Unstructured distributed	Passive Flood-based	-	Soft-state	-	-	-	-	-

Continued on next page

Table 2.1

Continued from previous page

SDPs	**Network type**	**Storage**	**Search**	**Description**	**Maintenance**	**Selection and usage**	**Scalability**	**Resource awareness**	**Mobility and fault tolerance**	**Security**
Konark [84]	Ad-hoc	Unstructured distributed	Active and passive Flood-based	XML	Passive Service adv. Event notification	SOAP	-	-	-	-
Allia [139]	Ad-hoc	Unstructured distributed	Active and passive Flood-based	-	Soft-state	-	Caching	-	Adjustment of adv. rate and radius	Authorization
GSD [55]	Mobile ad-hoc	Unstructured distributed	Active and passive Flood-based	XML (DAML+OIL)	Soft-state	-	Caching Service grouping	-	Adjustment of adv. rate and radius	-
Varshavsky et al. [158]	Mobile ad-hoc	Unstructured distributed	Active and passive	-	Service adv., hard state	Selection (client): Lowest hop count	Caching	-	Routing info. (DSR, DSDV)	-
Frank and Karl [74]	Mobile ad-hoc	Unstructured distributed	Active Flood based	-	Service adv., soft state, hard state	Selection (client): hop count	Caching	-	Routing info. (AODV)	-
INS [29]	Mobile	Unstructured distributed	Passive	Hierarchy of attribute-value	Service adv. Soft-state	-	Load distribution Service grouping	-	Compensation Late binding	-
Lenders et al. [108]	Mobile ad-hoc	Unstructured distributed	Active and passive Directed flow	-	Service adv., soft state	-	Caching Description aggregation	-	-	-
INS/ Twine [39]	Large and dynamic	Structured distributed (Flat)	Directed flow	Hierarchy of attribute-value	Hybrid state	-	Load distribution DHT	-	Compensation	-
Super string [140]	Wide-area	Structured distributed (Flat)	Directed flow	Hierarchical	-	-	-	-	Load distribution DHT	-

Continued on next page

Table 2.1

Continued from previous page

SDPs	Network type	Storage	Search	Description	Maintenance	Selection and usage	Scalability	Resource awareness	Mobility and fault tolerance	Security
CDS [75]	Mobile	Structured distributed (Flat)	Directed flow	Attribute-value	Soft-state	Selection (client): response time, database size	Load distribution	-	Compensation	-
One Ring [53]	Wide-area	Structured distributed (Flat)	Directed flow	Textual	Hard-state	Code binding	Caching DHT	-	Compensation	Authentication
SSDS [86]	Wide-area	Structured distributed (Hierarchical)	Directed flow	XML	Soft state	-	Hierarchical structure Load distribution Description aggregation	-	-	Authentication Authorization Confidentiality
CSP [107]	Wide-area	Structured distributed (Hierarchical)	Directed flow	Attribute-value	-	Selection (directory): distance, load, channel	Hierarchical structure Description aggregation	-	-	-
GloServ [34]	Local and wide-area	Structured distributed (Hierarchical)	Directed flow	XML (RDF schema)	Event notification Soft state	-	Hierarchical structure	-	-	Authentication
JXTA [155]	Wide-area	Structured distributed (Hybrid)	Hybrid	XML	Service adv. Soft state	-	Load distribution loose DHT	-	Compensation	Authentication Authorization Confidentiality
Service Rings [100]	Mobile ad-hoc	Structured distributed (Hybrid)	Directed flow	-	Polling	-	Structure optimization	-	Restructuring	-
LANES [99]	Mobile ad-hoc	Structured distributed (Hybrid)	Directed flow	-	Polling	-	Structure optimization	-	Compensation Restructuring	-

Continued on next page

Table 2.1

Continued from previous page

SDPs	**Network type**	**Storage**	**Search**	**Description**	**Maintenance**	**Selection and usage**	**Scalability**	**Resource awareness**	**Mobility and fault tolerance**	**Security**
Kozat & Tassiulas [102]	Mobile ad-hoc	Structured distributed (Hybrid)	Directed flow	-	Soft-state	-	Clustering Load distribution	-	Restructuring	-
Helmy [85]	Ad-hoc	Structured distributed (Hybrid)	Active and passive Hybrid	-	-	-	Clustering	-	Adjustment of zone radius	-
SD4WSN [4]	Heterogeneous, mobile sensor network	Structured distributed (Hybrid)	Active Hybrid	-	Hard-state	-	Clustering	Workload distribution	Clustering Reconfiguration	-

2.5 Conclusions

In this chapter, we describe the objectives of service discovery and we classify the existing SDPs according to the different methods conceived to achieve these objectives. Our classification takes into account the network type, the primary objectives of service discovery (storage, search, service description and maintenance) and the secondary objectives of SDPs (service selection and usage, scalability, resource awareness, mobility, dependability and security). Out of these objectives, we identify the *storage type* as being the dominant characteristic, since it directly influences the search method and the performance of SDPs.

Following the observations derived from the analysis of Table 2.1 and detailed in the previous section, we conclude that *resource awareness* is one research direction that is not sufficiently addressed by current state of the art. Resource awareness is especially important in the field of WSNs, where the constraint capabilities of the nodes rule out the existing discovery solutions. This thesis makes one step forward, by proposing an SDP particularly tailored to heterogeneous and dynamic WSNs. Since cluster-based solutions have the advantage that the knowledge can be distributed among the members of the clusters, depending on the hierarchical level, we design SD4WSN (described in Chapter 5), a service discovery protocol based on a simple and lightweight clustering structure, which allows a low maintenance overhead and a low discovery cost even in highly dense and dynamic sensor networks. For an informed decision regarding the clustering features required and the available solutions, we present a survey and classification of existing clustering algorithms in Chapter 3.

Chapter 3

A classification of clustering algorithms for wireless ad-hoc and sensor networks

Following the conclusion from Chapter 2 that clustering represents a method to handle heterogeneity in a resource-constraint environment, we are interested in a clustering solution that can be used to support service discovery in dynamic WSNs. Therefore, we present a survey and classification of clustering algorithms designed for wireless ad-hoc and sensor networks. Firstly, we introduce the clustering concept and the main reason for using it within this network environment. Secondly, we define the main classification criteria and describe each of the categories in turn. Thirdly, we provide an overview of the different clustering solutions, pointing out their characteristics and explaining the main construction steps. Fourthly, we give a comparative table that summarizes the characteristics of the various clustering algorithms according to the classification. We end this chapter with conclusions and open issues.

3.1 Preliminaries

Among many challenges faced by ad-hoc and sensor networks designers, *scalability* is a critical issue. The flat topology of these types of networks contains a large number of nodes that have to compete for the limited wireless bandwidth, handle sizeable routing tables and manage substantial traffic caused by network dynamics. One promising approach to solve the scalability problem is to abstract the network topology by building hierarchies of nodes. This process is commonly referred to as *clustering*.

We formally define the *clustering problem* following the definition given by Chen et al. [59]. We model the network as an undirected graph $G = (V, E)$, where V is the set of *vertices* or *nodes* and E is the set of edges or links that directly connect two nodes. The clustering process divides V into a collection of (not necessarily disjoint) subsets $\{V_1, V_2, ..., V_k\}$, where $V = \bigcup_{i=1}^{k} V_i$, such that each subset V_i induces a connected subgraph of G. Each such subset is a cluster. Typically, a particular vertex in each cluster, termed the *root* or *clusterhead*, is elected to represent the cluster.

To define a multi-level clustering hierarchy, we consider the abstracted graph $G^1 = (V^1, E^1)$, constructed from each clusterhead of $\{V_1, V_2, ..., V_k\}$ [59]. The set of nodes V^1 (representing the elected clusterheads from V) is regarded as the hierarchical *level 1*. There is an edge between two nodes $r_i, r_j \in V^1$ if and only if there is an edge of E from some node $v_i \in V_i$ to some node $v_j \in V_j$. The clustering process can be repeated for graph G^1, and subsequently for G^2, G^3 etc. Finally, a hierarchical *level k* of clusterheads can be obtained.

3.2 Classification

The classification that we propose provides a general overview of clustering design choices and attained performance. The classification criteria include the clustering purpose, assumptions, decision range, decision metrics, degree of mobility, resulting structure type, number of clusters and complexity. We analyse each criterion in turn and describe the associated categories.

3.2.1 Purpose

Clustering algorithms for ad-hoc and sensor networks improve network scalability by handling two important problems regarding the size and mobility of the network: they make a large network appear smaller, and a highly dynamic topology appear less dynamic [124]. Delay and message overhead represent the cost for clustering; these are further described in Section 3.2.9. In this section, we focus on the above described scalability improvements and show their direct benefits.

A large network appears smaller Grouping nodes into clusters leads to having restricted communication and data exchange, which improves on the following network operations:

- *Medium access control (MAC).* The access to the medium can be controlled and bandwidth can be allocated separately in each cluster, thus reducing the scope of inter-cluster interactions and avoiding redundant exchange of messages [114].
- *Routing.* The size of the routing tables is reduced by maintaining routes only to the clusterheads, and not to every node in the network [156].
- *Flooding.* The cost of flooding is reduced by decreasing the number of nodes that broadcast the message to only clusterheads and border nodes [148].
- *Data collection.* The data collected within a cluster is aggregated at the clusterhead and transmitted as a whole to the base station, thus avoiding excessive message exchange [83].
- *Service discovery.* The clusterheads maintain a service directory for nodes in their cluster. Thus, service discovery messages are transmitted only to the clusterhead nodes, and not in the whole network [4] (see Chapter 5).

A highly dynamic topology appears less dynamic Clustering can be used to partition the network with the objective of maintaining a relatively stable topology. This improves on the following network functionalities:

- *Routing.* Complete routing information is maintained only for intra-cluster routing. Intercluster routing is achieved by hiding the topology details within a cluster from external nodes, thus limiting far-reaching reactions to topology dynamics [124].
- *Collaborative processing.* Identifying nodes moving together and creating clusters based on joint movement allows for long-term intra-cluster collaborative processing (see Chapter 7).

Abstracting from the above specific purposes, several clustering algorithms are *generic* algorithms, meaning that they do not follow any particular objective, but rather propose a general solution that can be applied to various networking operations [148].

3.2.2 Assumptions

The general assumptions of clustering algorithms, unless otherwise stated in Section 3.3, are that the wireless communication is reliable (that can be achieved by using a reliable transport protocol [160]), and that the communication links are symmetrical. In addition, each clustering algorithm has a list of specific assumptions, based on the functionality that the lower layers of the communication stack (MAC, routing, transport) or other algorithms running on the nodes provide. Additional assumptions include the following:

- *Synchronization.* Clustering algorithms that require a series of coordinated phases among the network nodes assume the availability of a network synchronization mechanism [83].
- *Unique node IDs.* Weight-based clustering algorithms require unique IDs assigned to nodes, which can be used to break ties [43].
- *Localization.* Localization information represents the coordinates of the node location. This information is useful for grouping nodes based on their location [148].
- *Level of dynamics.* The level of dynamics, such as a generic stationary/mobile attribute or the concrete node speed is useful for reasoned cluster membership selection [58].
- *Global information.* The number of nodes within the network or the total remaining energy represent global information, which can be useful for achieving the desired clustering structure [83].
- *Routing information.* Routing tables may be needed to ease the communication among nodes during cluster organization [124].
- *Additional hardware capabilities.* Hardware capabilities can help achieve a better clustering structure by providing additional information about neighbouring nodes or improved communication abilities. Examples include the capability to measure the Received Signal Strength (RSSI) and the availability of multiple transmission power levels [83, 167].
- *Additional structures.* Additional structures such as spanning trees may facilitate the clustering process, but may also induce more overhead for maintenance [159].

- *Additional algorithms.* Additional algorithms include localized event detection, context-sharing, availability paths or distance between pairs of nodes. The output of these algorithms is semantic information used for clustering decisions [58].

Some of the above mentioned assumptions are in line with the decision metrics used to form clusters, such as unique node IDs or additional algorithms (see Section 3.2.3). Other assumptions are used to improve the clustering result by exploiting the availability of specialized hardware, or taking advantage of additional information, such as location or routing tables.

3.2.3 Decision metrics

The decision to become clusterhead or to join an existing cluster is typically based on the following metrics:

- *Time.* A node may become clusterhead on a time-dependant basis, i.e. if it is the first one in its neighbourhood that declares itself as clusterhead [76].

- *Probability.* A node may become clusterhead depending on a probabilistic measure. The probability is defined such that the desired number of clusterheads is reached without the need of global message exchange. The probability may depend on the number of nodes in the network, global aggregate energy, local residual energy, number of times the nodes has been clusterhead, cluster size, etc [83].

- *Weight.* A weight is an application-specific number assigned to every node in the network. The weight may depend on multiple measures, such as the node degree, distance to neighbours, movement speed, energy left, capability. The node ID is usually used to break ties. A node may become clusterhead if it has the highest weight among a group of nodes, depending on the decision range. Similarly, a node may choose to join the clusterhead with the highest weight [4]. Contrary to the probability metrics, weight metrics are deterministic.

- *Semantics.* Semantic properties refer to the relationship between pairs of nodes or among nodes in a group. Semantic properties include distance between nodes, availability paths between nodes, similar or relative mobility, location attribute or type of event detected. Clusters can be formed based on similar semantic properties of nodes [124].

The decision process may depend on more than one of the above metrics. For example, the clusterhead may be probabilistically selected, but the ordinary nodes choose a clusterhead based on a semantic property (e.g. the minimum distance to the neighbouring clusterheads) [83]. Similarly, nodes are grouped based on semantic information, but the clusterhead is chosen depending on the weight (see Chapter 7).

3.2.4 Decision range

The decision that each node takes is either autonomous, such that it does not depend on any other node in the network, or non-autonomous, where there are also other nodes that determine or influence the cluster membership. We denote this set of nodes with *the decision range.* The decision range can vary from as little as only 1-hop neighbours [4], to as large as the whole network [58].

3.2.5 Mobility

The design of a clustering algorithm depends on the degree of dynamics expected to be present in the wireless network. The network can be:

- *Mobile.* The clustering algorithm is designed to handle network mobility during any of its phases [43],[4].
- *Quasi-static.* The network is assumed to be static during the initial cluster setup phase. Strategies for cluster maintenance are given for the subsequent phases [44].
- *Static.* The network is static. Changes of topology rarely occur and do not represent the focus of the clustering algorithm [83].

A clustering algorithm designed for quasi-static or static networks has as main purpose to increase the scalability of the network with respect to the number of nodes. Algorithms that take mobility into account focus on reducing both the size and dynamics of the network (see Section 3.2.1).

3.2.6 Structure type

Given a graph $G = (V, E)$, we use the following notation:

- $\Gamma(v)$ is the open neighbourhood of v, $\Gamma(v) = \{u \in V \mid (u, v) \in E\}$;

- $\Gamma^+(v)$ is the closed neighbourhood of v, $\Gamma^+(v) = \Gamma(v) \cup \{v\}$;
- $\Gamma^+(S)$, where $S \subseteq V$ is a subset of V, is the set containing the closed neighbourhood of each $v \in S$, $\Gamma^+(S) = \bigcup_{v \in S} \Gamma^+(v)$;

Depending on its purpose, a clustering algorithm may: (1) partition the network into clusters, (2) select a set of nodes for the clusterhead role, or (3) achieve both partitioning and clusterhead selection. For the algorithms that achieve clusterhead selection, we summarize a number of definitions from graph theory [48, 59] that characterize the structure of clusterheads:

- *Dominating Set*: is a subset $S \subset V$ such that every vertex in $V \setminus S$ is adjacent to at least one node in S (see WCA [58]);
- *Independent Set*: is a subset $S \subset V$ such that no two vertices within the set S are adjacent in V (see Chapter 5);
- *Independent Dominating Set*: is a subset $S \subset V$, which is both a dominating set and an independent set (see DMAC [43]).
- *Connected Dominating Set*: is a subset $S \subset V$, which is a dominating set and induces a connected subgraph of G (see Wu and Li [165]).
- *Weakly Connected Dominating Set*: is a subset $S \subset V$, which is a dominating set such that $\Gamma^+(S)$ induces a connected subgraph of G (see Area [82]).
- *k-Dominating Set*: is a subset $S \subset V$ with the property that every node in G is at most k edges away from at least one of the nodes in S (see k-CONID [133]).
- *k-level hierarchy*: is a collection of subsets $\{V^1, V^2, ..., V^k\}$ of vertices corresponding to several levels of abstracted networks (see Section 3.1 and the algorithm proposed by Bandyopadhyay and Coyle [41]).

The dominating set structure types are mainly used to construct a backbone of clusterheads for restricted network flooding. The independent set is used for achieving a sparse set of clusterheads with the goal of lowering the overhead of intra-cluster communication. The k-level hierarchical structures are used for an organized, energy-efficient data collection.

3.2.7 Disjoint clusters

Depending whether a node may be part of one or more clusters, the output of the clustering algorithm falls in one of the following categories:

- *Disjoint clusters.* A node may belong to only one cluster [43].
- *Overlapping clusters.* A node may belong to more than one cluster [165].

Algorithms that partition the network into clusters and construct connected dominating sets of clusterheads have as result overlapping clusters. The reason is that the nodes that connect a set of clusters (gateway nodes) belong to all the adjacent connected clusters. Disjoint clusters are generally constructed when a node has to share a piece of information (such as id, sensed data, service offer) with the clusterhead. The clusterhead is thus responsible to make use of this information on behalf of the node.

3.2.8 Number and size of clusters

Since clustering improves the scalability of higher layer protocols by making a large network appear smaller (see Section 3.2.1), the number and size of clusters is an important metric in characterizing the performance of a given algorithm. However, when speaking about performance, it is important to relate to the application objectives. In some cases, it is desirable to have a small number of clusters (for example to route packets quickly between clusters), but in other cases it is important to keep the cluster size small and consequently form more clusters (for example to manage the structure in the presence of mobility).

Algorithms generate different cluster sizes, depending for example on the number of nodes in the network n, the average node degree D [43] or the probability p of becoming a clusterhead [83]. The number and size of clusters generated by semantic algorithms depend on the number of distinct semantic properties that represent clustering criteria. Algorithms that construct (weakly-)connected dominating sets usually use the *approximation factor* (the ratio between approximate and optimal solution) as a metric to characterize the performance of the algorithm [165].

3.2.9 Complexity

The complexity of a clustering algorithm is essential for estimating the latency and message overhead involved in building and maintaining the clusters.

To evaluate the *time complexity*, the algorithm is considered to start from a stable state. An event of a single, isolated change in this network (e.g. a link added or deleted) triggers a series of steps for restructuring the structure [45]. The time it takes the algorithm after this event to achieve a valid cluster structure is denoted as convergence time.

The *message complexity* defines the communication effort for creating and maintaining clusters [45]. For achieving minimum energy expenditure and processing load on the nodes, the overhead induced by clustering messages should be as low as possible.

3.3 Algorithm description

In what follows, we briefly describe the state of the art in clustering algorithms for wireless ad-hoc and sensor networks. We give a logical structure to this section by grouping the algorithms based on the decision metrics, as this criteria highlights the main methods used for clustering in this networking environment.

3.3.1 Decision based on weights

Weight-based clustering algorithms assign each node in the network an application specific measure, which usually includes a number of parameters related to how suitable the node is for the clusterhead role [133]. The election of clusterheads involves (1) the dissemination of every node's weight to a group of nodes in the network that represent the decision range, and (2) the comparison of these weights and the selection of the best node as clusterhead in a greedy manner.

The LCA clustering algorithm [69] was one of the first weight-based algorithms designed for ad-hoc networks. LCA functions together with a TDMA MAC protocol for discovering the one-hop and two-hop connectivity information and to determine the bidirectional links. Each node selects as its own clusterhead the neighbouring clusterhead with the lowest ID to which it is bidirectionally connected. A node that can hear two or more clusterheads is a gateway, otherwise, a node is an ordinary node.

With the DCA clustering algorithm [114, 44], the decision is based only on the IDs of the 1-hop neighbours. The algorithm is divided in two phases: the cluster formation phase and the maintenance phase. During the cluster formation phase, nodes with the higher IDs in their neighbourhood are selected as clusterheads. The rest of the nodes in the network are assigned to the existing clusterhead nodes, forming thus disjoint clusters. The network topology should not change during this phase. After the cluster formation phase, the clusterhead nodes constitute an independent dominating set. During the maintenance phase, the clustering structure is reconfigured as a result of node mobility or failure.

DMAC [43] is a clustering algorithm particularly designed for mobile networks. Similar to DCA, nodes decide their role depending on the one-hop neighbourhood information. The difference is that DMAC does not perform in separate phases. Each node reacts locally to any variation in the surrounding topology, changing its role accordingly. The variation of the topology is represented by addition and deletion of links to neighbouring nodes. The node with the highest weight among its unassigned neighbours declares itself as clusterhead. The rest of the nodes choose as clusterhead the neighbour with the highest weight. When a link breaks down between a node and its clusterhead, the node has to elect a different clusterhead from its neighbourhood. A new link between two nodes is handled as follows: if a node notices the presence of a new neighbour clusterhead with larger weight, it will join the new clusterhead. Similarly, if a clusterhead sees a new neighbouring clusterhead with a higher weight, it will give up its role and affiliate with the new neighbour.

A generalization of DMAC is G-DMAC [42], where a newly initialized node joins the clusterhead with the highest weight in its one-hop neighbourhood (similar to DMAC). While the topology changes, however, the node remains member of this clusterhead v as long as there is no other neighbouring clusterhead u with weight $w(u) > w(h) + h$, given the parameter $h \geq 0$. Another parameter k is introduced that defines the maximum number of clusterhead neighbours that a clusterhead is allowed to have.

Other protocols base their election decisions on complete information over a number of hops or even from the whole network. The Max-Min D-Cluster algorithm [32] uses the d-hop information for clusterhead election. Each node initiates two rounds of flooding over d hops for building the cluster membership. During the first round of flooding, the largest node ID is propagated in each node's d-neighbourhood. At the end of this phase, the surviving node IDs are elected clusterheads. The second round of flooding propagates the smaller surviving node IDs over d hops. The smallest node ID appearing in both flooding stages is chosen as clusterhead by the other nodes. When the election algorithm finishes, nodes are at most d hops away from the clusterhead.

The k-CONID algorithm (k-hop connectivity ID) [133] constructs a k-dominating set of clusterhead nodes, by assigning each node a weight computed from the connectivity degree and the ID of the node. The algorithm is initiated by a flooding request for clustering to all the nodes in the network. All the nodes whose weights are the largest among all their k-hop neighbours become clusterheads. The other nodes choose a k-hop neighbouring clusterhead with the largest weight as clusterhead. Each node broadcasts its decisions after all its k-hop neighbours with larger weights have already done so. This algorithm constructs overlapping clusters, where the nodes that belong to more than one cluster are gateway nodes.

The WCA algorithm proposed by Chatterjee et al. [58] takes into account the node degree, transmission power, battery power and the speed of the nodes, for achieving the optimal operation of the MAC protocol. Each node calculates a combined weight from these parameters, which is then disseminated in the whole network. The node with the global minimum weight is chosen as clusterhead. This node and its neighbours are excluded from subsequent clustering decisions. The process is repeated until all the nodes are clustered.

Algorithms that construct connected dominating sets focus mainly on clusterhead selection. In the algorithm proposed by Wu and Li [165], every node exchanges its neighbour set with all its neighbours. If a node has two unconnected neighbours, it declares itself as clusterhead. To reduce the size of the connected dominating set, additional rules are introduced, where certain clusterhead nodes give up their roles. The decision is based on a greedy selection of the lowest node ID.

Stojmenovic et al. [148] improve the algorithm proposed by Wu and Li by having each node assigned a weight, computed based on the node degree and the x and y absolute coordinates. The rules for reducing the connected dominating set are now based on the node degree and location.

Wan et al. [159] propose an algorithm for the construction of a connected dominating set, by assuming the existence of a rooted spanning tree structure already in place. The weight of a node is given by the ordered pair of the tree level and the node ID. The algorithm consists of two phases: the construction of a maximal independent set and the construction of a dominating tree, whose internal nodes become a connected dominating set.

The Area clustering algorithm [82] constructs a weakly connected dominating set by using as weights the ordered pairs of node degree and node ID. A node becomes clusterhead if it has the highest weight among its neighbours that are not yet members of other clusters. The weakly connected dominating set is composed of the clusterhead nodes and a selected set of border nodes.

The C4SD clustering algorithm (see Chapter 5) is specifically designed for mobile networks, where the weight is represented by the node capability and degree of dynamics. Every node in the network chooses as parent the neighbour with the highest capability grade. If such a node does not exist, the node itself is a clusterhead. The output of the algorithm is represented by disjoint clusters, whose clusterheads form an independent set. A detailed description of C4SD is found in the next chapters.

3.3.2 Decision based on time

Time-based algorithms enable the election of clusterhead nodes based on the order in time when the candidate nodes announce their candidacy. For example, the order in occupying the shared wireless medium can determine how suitable are nodes for the clusterhead role. Following this idea, the Passive Clustering algorithm [76] uses the *first declaration wins* rule. This rule specifies that the node that sends a packet first becomes the clusterhead for the rest of the nodes in its one-hop neighbourhood.

The ACE clustering algorithm [56] makes its decisions on a combination of time and weight methods. The weight represents the number of loyal followers, e.g. the number of unclustered neighbours. An unclustered node spawns a new cluster by declaring itself as clusterhead whenever it finds that it can gain at least a number of minimum loyal followers, which is a function that depends on the time that passed since the protocol was initiated for that node.

3.3.3 Probabilistic decision

Probabilistic protocols are designed mainly for static networks (e.g. static Wireless Sensor Networks), as they run in synchronized rounds for balancing the energy consumption. Each round is composed of a number of phases, which are typically the cluster formation phase and the steady phase. During the cluster formation phase, a different set of clusterheads is elected based on a certain probability. This probability depends on the desired characteristics of the resulting clustering structure. The probabilistic decisions are usually combined with weight-based decisions: the clusterheads are elected probabilistically, but the rest of the nodes choose their clusterhead based on weights or semantic information (e.g. minimum distance to the clusterhead, link quality, etc). During the steady phase, sensor data is transmitted to the clusterheads, which in turn send it to the base station.

The source of inspiration for many protocols using probabilistic decisions is LEACH [83]. With LEACH, the probability of a node to become clusterhead is a function of the desired number of clusterheads in the network and the number of times a node has already been a clusterhead. When deciding to which cluster they belong, nodes choose the clusterheads which are physically closer.

The HEED clustering algorithm [167] computes the probability based on the estimated residual energy of the node, the reference maximum energy and the desired percentage of clusterheads in the network. A node chooses as clusterhead the one with the lowest intracluster communication cost.

The MOCA algorithm [168] constructs a k-dominating set of clusterhead nodes. A node becomes clusterhead based on a probability determined a priori depending on the network size. A clusterhead advertisement is forwarded k hops away. A node that receives such an advertisement joins the cluster even if it already belongs to another cluster, becoming thus a boundary node.

Bandyopadhyay and Coyle [41] propose an algorithm that constructs a h-level hierarchical clustering structure. The algorithm consecutively elects the clusterheads starting from level 1 and ending with level h, using probabilities p_1,..., p_h. The structure on each level l is a k_l-dominating set. The parameters p_l and k_l are computed such that the total energy consumption is minimized.

The EEMC algorithm [91] constructs a similar k-level hierarchical structure. Each node is assumed to know its location coordinates. At the beginning of the cluster formation phase, the sink node collects the location information from the nodes and sends back the total remaining energy of the network and the total distance between the nodes and the sink. The probability for the first hierarchical level of clusterheads is computed using these two values. For the next level, the clusterheads broadcast the total remaining energy and the total inter-node distance in their cluster, which are used to compute the next level probability.

3.3.4 Decision based on semantic information

McDonald and Znati [124] describe an (α, t) clustering algorithm designed to support routing in large ad-hoc networks, taking the node mobility as the criteria for cluster organization. The cluster internal paths are expected to be available for a period of time t with a probability of at least α. A node can join a cluster if all the destinations within the cluster are reachable via (α, t) paths. If a node is unable to join a cluster, it will create its own orphan cluster. Within a cluster, nodes maintain topology information and routes to every cluster destination.

Bouhafs et al. [49] propose a clustering algorithm based on semantic information. A user query is disseminated through the network looking for a specific group of sensor nodes. When the query reaches a node that satisfies it, this node becomes a clusterhead and starts forming a cluster that contains all nodes in its region that satisfy the same query.

Wang et al. [162] proposes an algorithm that combines semantic and weight decision metrics. The main clustering criterion is the semantic information represented by the location attribute. The clustering process is started by a node that sends a flooding message to the whole network. Nodes that hear this message wait for an amount of time proportional to their energy level. After the waiting period, nodes announce the intention of becoming clusterhead candidates at their particular location through a broadcast, which contains the energy level information. Other candidates that are present on the same location and hear such a broadcast cancel their timers and rebroadcast the higher energy. Ties are broken through the node IDs.

In the Smart Clustering algorithm [150], nodes are clustered according to their physical relationships, such as *objects on a table*. A master node (the table) senses the objects placed on it and automatically integrates them in its cluster.

Tandem (see Chapter 7) is an algorithm for spontaneous clustering of mobile wireless sensor nodes facing similar context (such as moving together). Tandem assumes that each node runs a shared-context recognition algorithm, which provides a number on a scale, representing the confidence value that two nodes are together. Each node periodically computes the confidence of sharing the same context with its neighbours. The selection of clusterheads is weight-based: the node with the highest weight among its neighbours with which it shares a common context declares itself as clusterhead. A regular node subscribes to the clusterhead with which it shares a common context and has the highest weight.

3.4 Comparative table

Table 3.1 shows a comparison among the various clustering algorithms presented above. Similarly to Section 3.3, we group the algorithms based on the decision metrics. Dashes present in the "Purpose" or "Assumptions" fields correspond to algorithms that are generic or which do not have any additional assumptions. Dashes within the "Number of clusters" or "Complexity" fields correspond to lack of information or systematic analysis of the algorithms. The remaining dashes correspond to "not applicable" features.

A close analysis of Table 3.1 exposes the following interconnections among various classification criteria:

- *Purpose vs. decision metrics.* Probabilistic algorithms are generally used for data collection in wireless sensor networks, due to the need of a rotation scheme among clusterhead nodes that facilitates load balancing. Networking operations, such as MAC or routing, can be improved by using weight-based clustering algorithms, where the clusterhead node is chosen in a greedy manner, being the best among a set of candidates. Semantic information can be used for a more informed decision regarding the appropriate clustering structure, and thus relates to different objectives, from routing to collaborative processing.

- *Assumptions vs. decision metrics.* Weight-based algorithms require a strict order among nodes, determined by the assigned weights. The node ID is generally used to break ties between nodes with the same weights. Therefore, unique node IDs is a common assumption for weight-based clustering algorithms. Probabilistic decisions are typically used when rotation of clusterhead role is necessary for balancing energy consumption among nodes in the network. Rotation of clusterheads can be done when all the nodes are participating at the same time in the election process. Hence, the election of clusterheads takes place at the beginning of each synchronous round. A common assumption of probabilistic algorithms is therefore the existence of a synchronization mechanism among the network nodes. Decisions based on semantic information require additional algorithms or information (such as location or routing tables), from which the semantic knowledge used for clustering is derived.

- *Mobility vs. assumptions and decision metrics.* Within static networks, the assumption of synchronous rounds is easier to be achieved than in mobile networks. Therefore, synchronization is typically found as assumption for probabilistic algorithms designed for static networks. Algorithms for mobile networks use weights and semantic information, since these are flexible metrics that can incorporate knowledge about node speed or level of dynamics.

- *Number of clusters vs. decision metrics.* Probabilistic algorithms control the number of clusters by setting the probability of becoming a clusterhead according to the application requirements. The number of clusters resulting from semantic algorithms depend on the number of semantic properties existing in the network, or on the number of nodes that have similar or distinct characteristics (i.e. moving together or separately).

- *Complexity vs. decision metrics.* The time complexity of the algorithms based on probabilistic decisions can be as low as $O(1)$. The reason is that the decision of a node is independent of the decisions of other nodes. Algorithms based on weights or semantic information have higher time complexities, typically at least $O(n)$.

By analysing Table 3.1 following the classification criteria, we observe that only few algorithms are designed to handle mobility. In the following, we describe the reasons why existing algorithms are less suitable for mobile environments. Firstly, electing the clusterheads based on information from nodes which are multiple hops away leads to high overhead and slow reaction to topology changes. Secondly, maintaining complete intra-cluster information is an expensive task which results in a high traffic. Thirdly, the complexity of the multi-layer clustering algorithms leads to a lot of effort in building and maintaining the desired structure.

We conclude that the majority of algorithms focus on improving the scalability of large static networks, rather than addressing mobility within these networks. In the following section, we analyse future research directions derived from this observation.

Table 3.1: Overview of clustering algorithms

Algorithm	Purpose	Assumptions	Decision metrics	Decision neigh. range	Mobility	Clustering structure type	Disjoint clusters	Number of clusters	Complexity
LCA [69]	MAC, routing, flooding	Unique node ID	Weight (node ID)	2-hops	Quasi-static	Independent dominating set	No	-	Time: $O(n)$ [28]
DCA [114, 44]	MAC, routing	Unique node ID	Weight (node ID)	1-hop	Quasi-static	Independent dominating set	Yes	$n/(1+D/2)$ [45, 44]	Time, msg.: $O(n)$ [114, 44]
DMAC [43]	Routing	Unique node ID	Weight (node ID)	1-hop	Mobile	Independent dominating set	Yes	$n/(1+D/2)$ [45]	Time, msg.: $O(n)$ [92, 43]
G-DMAC [42]	MAC, routing	Unique node ID	Weight (node ID)	1-hop	Mobile	Dominating set	Yes	-	Time: $O(2n/(k+2))$ [92]
Max-Min D-Cluster [32]	Routing	Synchronous rounds Unique node ID	Weight (node ID)	d-hops	Static	d-dominating set	Yes	-	Time: $O(d)$ [32]
k-CONID [133]	Routing	Unique node ID, algorithm initiated by flooding	Weight (node degree and ID)	k hops	Quasi-static	k-dominating set	No	-	-
WCA [58]	MAC	Distance to neigh., node speed	Weight (node degree, distance to neigh., speed, cumulative time as a clusterhead)	Network wide	Mobile	Dominating set	Yes	-	Time, msg.: $O(n^2)$ [166]
Wu and Li [165]	Routing	Unique node ID	Weight (connectivity, node ID)	2 hops	Mobile	Connected dominating set	No	Approx. factor: $n/2$ [159]	Time: $O(n^3)$ Msg.: $O(n^2)$ [159]
Stojmenovic et al. [148]	-	Node location	Weight (node degree, x and y coordinates)	2 hops	Static	Connected dominating set	No	Approx. factor: $n/2$, n[159]	Time: $\Omega(n)$ Msg.: $O(n^2)$ [159]

Continued on next page

Table 3.1

Continued from previous page

Algorithm	**Purpose**	**Assumptions**	**Decision metrics**	**Decision neigh. range**	**Mobility**	**Clustering structure type**	**Disjoint clusters**	**Number of clusters**	**Complexity**
Wan et al. [159]	Routing	Spanning tree constructed, synchronous rounds	Weight (tree level, node ID)	1 hop	Static	Connected dominating set	No	Approx. factor: ≤ 8 [159]	Time: $O(n)$ Msg.: $O(n\log n)$ [159]
Area [82]	Routing	Unique node ID	Weight (node degree, node ID)	3 hops	Static	Weakly-connected dominating set	Yes	Approx. factor: ≤ 110[82]	Time, msg.: $O(n)$ [82]
C4SD [4]	Service discovery	Unique node ID, degree of dynamics	Weight (node capability and dynamics, node ID)	1-hop	Mobile	Independent set	Yes	$\frac{n}{D}(1-e^{-D})$	Time, msg.: $O(n)$
ACE [56]	-	-	Time, weight (number of loyal followers)	2-hops	Static	2-dominating set	No	-	Time: $O(d)$ [56]
Passive Clustering [76]	Routing	-	Time	1-hop	Quasi-static	Independent dominating set	No	-	-
LEACH [83]	Data collection	Aggregate network energy, RSSI, synchronous rounds	Probabilistic	Autonomous and 1-hop	Static	Dominating set	Yes	k [83]	Time: $O(1)$ [28]
HEED [167]	-	Multiple power levels, synchronous rounds	Probabilistic	Autonomous and 1-hop	Static	Dominating set	Yes	-	Time: $O(1)$ Msg.: $O(n)$ [167]
MOCA [168]	Data collection	Unique node ID	Probabilistic	k-hops	Static	k-dominating set	No	-	Time: $O(k)$ [168]
Bandyopadhyay and Coyle [41]	-	Synchronous rounds	Probabilistic	max(k_1, k_2, ..., k_h)	Static	h-level hierarchical clustering	Yes	Level i: $(1-p_i)/p_i$ [41]	Time: $O(k_1+k_2+\ldots+k_h)$ [41]

Continued on next page

Table 3.1

Continued from previous page

Algorithm	**Purpose**	**Assumptions**	**Decision metrics**	**Decision neigh. range**	**Mobility**	**Clustering structure type**	**Disjoint clusters**	**Number of clusters**	**Complexity**
EEMC [91]	Data collection	Node location, synchronous rounds	Probabilistic	Node-sink and 1 hop	Static	k-level hierarchical clustering	Yes	Level i: $O(n^{\frac{1}{2}+..+\frac{1}{2^i}})$ [91]	Time: $O(\log\log n)$ Msg.:$O(n)$ [91]
(α, t) [124]	Routing	Intra-cluster and inter-cluster routing tables	Semantic ((α, t) availability paths)	Neigh. clusters	Mobile	-	Yes	Num. of mobile groups	-
Bouhafs et al. [49]	Data collection	-	Semantic: satisfy query	Network wide	Static	-	No	Num. query types	-
Wang et al. [162]	Information dissemination	Location attribute	Semantic (location attribute) and weight (energy, node ID)	Network wide	Quasi-static	2-dominating set	Yes	$\geq$ num. location attributes	Time, msg.: $O(n)$ [162]
Smart Clustering [150]	-	Event detection algorithm	Semantic (location)	1-hop	Quasi-static	Dominating set	Yes	Num. locations	-
Tandem [6]	Collaborative processing	Context-sharing algorithm, unique weight	Semantic (joint movement), weight	1-hop	Mobile	Dominating set	Yes	Num. context-sharing groups	Time: $O(n)$

3.5 Conclusions

Clustering algorithms for wireless ad-hoc and sensor networks aim to improve the scalability of such networks, by creating virtual overlays that make networks appear smaller and less dynamic [124]. We have defined a classification that tries to capture the essential characteristics of clustering algorithms, which differentiates them in terms of design decision and performance. The design depends on the specific purpose, assumptions, decision range and metrics, degree of dynamics and desired structure type. In particular, the decision metrics is a classification criterion that identifies the basic clustering mechanism used. The performance is governed by the time and message complexity to build and maintain the clusters, and by metrics such as the number of clusters.

As pointed out in Section 3.4, mobility needs to be explored further, possibly by using movement sensors integrated with each wireless node. This would facilitate the extraction of mobility patterns and identification of clusters based on joint movement, and thus adapt the clustering procedure by including context relevant information.

In the next chapter, we make a step forward by proposing a generalized framework for weight-based algorithms designed for mobile networks that make localized decisions (1-hop neighbourhood) and produce disjoint clusters. From Table 3.1 we derive that DMAC, G-DMAC, C4SD and Tandem algorithms fall in this category. C4SD and Tandem represent our contribution to clustering in dynamic environments, and are detailed in Chapters 5 and 7. These algorithms are designed to support energy-efficient service discovery and collaborative processing respectively, for the wireless sensor network environment.

Chapter 4

A generalized clustering algorithm for dynamic wireless sensor networks

Based on the classification presented in Chapter 3, we propose a general clustering algorithm for dynamic sensor networks, that makes localized decisions (1-hop neighbourhood) and produces disjoint clusters. The purpose is to extract and emphasise the essential clustering mechanisms common for a set of particular algorithms, which allows for a better understanding of these algorithms and facilitates the definition and demonstration of common properties.

4.1 Generalized clustering algorithm

The definition of the clustering problem from Chapter 3 describes clustering as the division of the set of vertices or nodes V into a collection of not necessarily disjoint subsets. In this chapter, we refer to a particular case of the clustering problem, where clusters are *disjoint* subsets of V.

We assume that each node is assigned a unique *weight*, which is an application-specific number that can be calculated based on different metrics, such as the degree of dynamics, the resource availability, the battery level etc. The node hardware identifier may be used to break ties. In this way, we abstract from the physical characteristics of nodes by only using the *weight* measure in the algorithm description.

In what follows, we present the input and the output of the generalized clustering algorithm and then we proceed with the algorithm description. Table 4.1 from the Appendix presents a summary of notation used in this chapter, which is introduced and explained within the next sections.

4.1.1 Input

The clustering structure is constructed based on the decision of each node to select a certain *parent*. The decision of each node v depends on the *neighbourhood information*, which includes the set neighbours (the nodes connected through links with v), the weights of the neighbours, the semantic relationship between v and its neighbours, and the current state of the neighbours (their clusterheads), which becomes known to v during the algorithm operation.

The input of the algorithm for each node v is the following:

- $\boldsymbol{\Gamma(v)}$, the open neighbourhood of v, $\Gamma(v) = \{u \in V \mid (u, v) \in E\}$;
- $\boldsymbol{\Gamma^+(v)}$, the closed neighbourhood of v, $\Gamma^+(v) = \Gamma(v) \cup \{v\}$;
- $\boldsymbol{w(u), \forall u \in \Gamma^+(v)}$, the weight of v and the weights of all the neighbours of v.
- $\boldsymbol{s(v, u), \forall u \in \Gamma(v)}$, the semantic relationship between node v and all its neighbours; $s(v, u)$ equals 1 if v and u have similar semantic properties and 0 otherwise.

We remark that the clustering algorithm runs on a *dynamic* network, where the position and the semantic relationships among pairs of nodes change in time, and therefore, the input of the algorithm varies accordingly.

4.1.2 Output

The output of the generalized clustering algorithm is a set of disjoint clusters, where for each cluster there is a *root* or *clusterhead* node, selected to represent the cluster. To achieve this, each node selects a *parent* from its set of neighbours. The parent of the clusterhead node is the node itself. The weights of the nodes are used in the parent selection: the higher the weight of a neighbour, the more chances to be selected as parent. However, the parent does not always have a higher weight than the node: the weight is used in the decision process, but also other factors contribute to the parent selection (e.g. whether the neighbour is already a clusterhead). The parent of node v is used for communication between v and its clusterhead. A node v may be *unassigned*, if it cannot cluster with any of its neighbours. In this case, the node does not have any parent or root.

For each node v in the network, the output of the algorithm is the following:

- $\boldsymbol{p(v)}$, the parent of v; $p(v) \in V \cup \{\perp\}$.

  We make the following observations:

  - If $p(v) \in V$, then v is called *assigned*. Otherwise, if $p(v) = \perp$, then v is called *unassigned*.
  - If $p(v) = v$, then v is called *root* or *clusterhead*.

- $\boldsymbol{r(v)}$, the root or clusterhead of v; $r(v) \in V \cup \{\perp\}$.

  We make the observation that $p(v) = \perp \Longrightarrow r(v) = \perp$.

This output describes a *feasible* clustering structure if the directed graph $G_p = (V_p, E_p)$, defined by $V_p = \{v \in V \mid p(v) \neq \perp\}$ and $E_p = \{(v, u) \mid u = p(v) \wedge u \neq v\}$, is a forest of routed trees where the root of each node v is a node equal to the root of its parent (i.e. $\forall v \in V_p$, we have $r(v) \in V_p$ and $r(v) = r(p(v))$). In this way, each tree in G_p forms a cluster with the root of the tree as clusterhead. The nodes in G that are not part of G_p are unassigned.

4.1.3 Properties

Due to the network dynamics, the output of the clustering algorithm actively has to adapt to the input changes. This adaptation is described in detail in Section 4.1.4. For static networks (i.e. networks with stable topology and semantic relationships), we require the following properties that a clustering algorithm has to fulfil:

Property 1 *The algorithm produces disjoint clusters.*

Property 2 *Each cluster is organized as a tree, following the parent-children relationship.*

Property 3 *The structure stabilizes to a feasible clustering structure after a finite number of rounds.*

Note that Properties 1 and 2 also hold for dynamic networks. For the definition of a *round*, see Section 4.1.4.

4.1.4 Description

The generalized clustering algorithm follows the $\mathcal{LOCAL}$ message passing model, where global structures are constructed based on local information and using local message exchange [132]. Following this model, each node in the network performs some computations and communicates only to its direct neighbours by exchanging messages based on rounds.

The algorithm is based on the following assumptions:

Assumption 1 *Each weight is unique (the node hardware ID can be used to break ties).*

Assumption 2 *The wireless communication is reliable (this can be achieved by using a reliable transport protocol [160]).*

Assumption 3 *The communication links are symmetrical, i.e.* $\forall v \in V$, $\forall u \in \Gamma(v)$, $v \in \Gamma(u)$ *(asymmetric links can be hidden by the MAC protocol).*

Assumption 4 *The semantic relationship is symmetrical, i.e.* $\forall v, u \in V$, $s(v, u) = s(u, v)$ *(if two neighbours have different views on the semantic relationship between them, thay can reach an agreement, e.g.* $s(v, u)$ *is given by the node with the highest weight).*

Assumption 5 *Each node is aware of its neighbours, the weights of the neighbours and the semantic relationship with the neighbours, representing the input of the algorithm (see Section 4.1.1).*

To decide on the clustering structure, each node selects one of its neighbours as parent. A node is a clusterhead if it is its own parent. A node that is not a parent is called a *leaf* node. The *height* of the cluster is the longest path from the root node to a leaf. The identity of the clusterhead is transmitted from parents to children. Therefore, the parent selection is enough to uniquely determine the cluster membership: each node learns from its parent the root of the cluster.

The parent may be selected either periodically or on demand, as a result of a change in the network topology, a reception of update information from neighbours or a change in the semantic relationship with the neighbours. For the sake of simplicity, we consider that the parent is selected periodically. We define a *round* as the time between two consecutive parent selections. One round is long enough in order for all the messages sent by a node during a parent selection to be received by its neighbours (by Assumption 2). We make the observation that nodes are not required to be synchronized: rounds are a tool for the proofs only.

The decision of parent selection is based on the local neighbourhood information: the input of the algorithm (the weight of the node, the set of neighbours and their weights and the semantic relationship with the neighbours) and the current state of the neighbours (represented by the root node of each neighbour).

For each vertex, two subsets of neighbours are given:

- $\boldsymbol{N_1(v)} \subseteq \Gamma^+(v)$ is the subset of neighbours with which node v may be in a common cluster (the decision may be dependent on the semantic relationships among neighbours or other parameters);
- $\boldsymbol{N_2(v)} \subseteq N_1(v)$ is the subset of $N_1(v)$, representing the nodes that are eligible to become parents of v; if the algorithm assigns a parent to v, it either chooses it from the set $N_2(v)$ or it assigns the node itself as parent (in this case, v becomes clusterhead).

Furthermore, three different conditions are given, to be used in the decision process of vertex v:

- $\boldsymbol{P_1(v)}$ is the condition on which node v chooses a different parent (i.e. if the algorithm reaches condition $P_1(v)$ (line 8 of Algorithm 1) and $P_1(v) = \textit{true}$, then node v changes the parent to the best candidate from the set $N_2(v)$).
- $\boldsymbol{P_2(v)}$ is the condition on which node v becomes root (i.e. if the algorithm reaches condition $P_2(v)$ (line 13 of Algorithm 1) and $P_2(v) = \textit{true}$, then node v becomes root by assigning $p(v) = v$).
- $\boldsymbol{P_3(v)}$ is the condition on which node v informs the neighbours that one of them has to resign from the clusterhead role (i.e. if $P_3(v) = \textit{true}$, then v sends a *Resign* message to the neighbours).

In the following, we pin down two particular neighbours of v that have specific roles in the algorithm execution:

- $\boldsymbol{y(v)}$ is the best candidate for the parent role: $y(v)$ is the neighbour of v with the highest weight in $N_2(v)$, i.e. $w(y(v)) = max\{w(u) \mid u \in N_2(v)\}$; based on Assumption 1, $y(v)$ is uniquely determined; $y(v)$ is chosen as parent if the algorithm reaches condition $P_1(v)$ and $P_1(v) = \textit{true}$;

- $\boldsymbol{z(v)}$ is the neighbour of v with the highest weight that has to resign from the clusterhead role, based on condition $P_3(v)$; when node v sends a *Resign* message to the neighbours with parameter $w(z(v))$, all the neighbours with smaller or equal weights than $z(v)$ have to resign from the clusterhead role and search for a new parent.

Algorithm 1 formally describes the generalized algorithm. Each node, when it powers up, enter an *Initialization* phase, where the local variables are initialized and the node becomes unassigned (i.e. $p(v) = r(v) = \perp$). Then, the selection of a parent is done on a periodic basis, as described earlier, by calling the *SelectParent* function. The structure of the *SelectParent* function is the following:

1. Update the local information from the input given by the lower layers of the communication stack, such as the MAC (line 2). From this information, each node v selects the subset of neighbours $N_1(v)$ which are eligible to be part of the same cluster as v. Then v builds the set $N_2(v)$ from the neighbours in $N_1(v)$ which can become parents.

2. Decide the parent, and consequently the root node, based on $N_1(v)$, $N_2(v)$, $P_1(v)$ and $P_2(v)$ (lines 3-21). The decision process can be described as follows:

   (a) In case the set $N_1(v)$ is empty, node v becomes unassigned (lines 3-5).

   (b) Otherwise, if $N_2(v)$ is not empty and $P_1(v) =$ *true*, the node with the highest weight from $N_2(v)$ (i.e. $y(v)$) becomes the parent of v (lines 7-11).

   (c) If $N_2(v)$ is empty, v is not root and $P_2(v) =$ *true*, node v becomes clusterhead (lines 12-15).

   (d) If $N_2(v)$ is empty, v is not root and $P_2(v) =$ *false*, node v becomes unassigned (lines 16-19).

3. If the root has changed or there is a new node in the neighbourhood, inform the neighbours about the current root, by sending a *SetRoot* message (lines 22-24).

4. Based on condition P_3, inform the neighbours that one of them has to resign from the clusterhead role, by sending the message *Resign* (lines 25-27).

The algorithm uses two types of messages, *SetRoot* and *Resign*. When v receives a *SetRoot* message from its parent, it learns the root of its cluster and resends the message, so that the children of v also get informed about their root. Upon receiving a *Resign* message with parameter w, if v is a root node with a lower or equal weight to w, then it has to give up its role and search for a new parent. The neighbour from $N_2(v)$ with the highest weight, $y(v)$, becomes the new parent.

In the following, we describe four special cases of Algorithm 1, by giving concrete definitions to the sets $N_1(v)$ and $N_2(v)$ and to conditions $P_1(v), P_2(v)$ and $P_3(v)$.

4.1.5 Special cases

The weight-based algorithms C4SD (see Chapter 5), Tandem (see Chapter 7), DMAC [43] and G-DMAC [42] are considered special cases of Algorithm 1. The definitions of the sets $N_1(v)$ and $N_2(v)$ and conditions $P_1(v), P_2(v)$ and $P_3(v)$ for each of these algorithms are the following:

- **C4SD**:
  - $N_1(v) = \Gamma^+(v)$
  - $N_2(v) = \{u \in N_1(v) \mid w(u) > w(v)\}$
  - $P_1(v),\ P_2(v) : true$
  - $P_3(v) : false$
- **DMAC**:
  - $N_1(v) = \Gamma^+(v)$
  - $N_2(v) = \{u \in N_1(v) \mid w(u) > w(v) \wedge r(u) = u\}$
  - $P_1(v),\ P_2(v) : true$
  - $P_3(v) : false$

  By restricting the set $N_2(v)$ to include only root nodes, DMAC generates one-hop clusters (i.e. each assigned node is either a clusterhead or its parent is a clusterhead).
- **Tandem**:
  - $N_1(v) = \{u \in \Gamma(v) \mid s(v, u) = 1\}$

Algorithm 1: Generalized clustering algorithm - node v

```
Initialization:
 1. $r(v) \leftarrow \perp$; $p(v) \leftarrow \perp$; $r(u) \leftarrow \perp,\ \forall u \in \Gamma(v)$

SelectParent: // Build the clustering structure by selecting the parent
 1. $r_0 \leftarrow r(v),\ \Gamma_0(v) \leftarrow \Gamma(v)$
 2. Update $\Gamma(v),\ \Gamma^+(v),\ N_1(v),\ N_2(v),\ y(v)$.
 3. if $N_1(v) = \emptyset$ then
 4.     $p(v) \leftarrow \perp$
 5.     $r(v) \leftarrow \perp$
 6. else
 7.     if $N_2(v) \neq \emptyset$ then
 8.         if $P_1(v)$ then
 9.             $p(v) \leftarrow y(v)$
10.             $r(v) \leftarrow r(p(v))$
11.         end if
12.     else if $(p(v) \neq v)$ then
13.         if $P_2(v)$ then
14.             $p(v) \leftarrow v$
15.             $r(v) \leftarrow v$
16.         else
17.             $p(v) \leftarrow \perp$
18.             $r(v) \leftarrow \perp$
19.         end if
20.     end if
21. end if
22. if $(r(v) \neq r_0) \vee (\Gamma(v) \setminus \Gamma_0(v) \neq \emptyset)$ then
23.     Send SetRoot(v,r(v)) to neighbours.
24. end if
25. if $P_3(v)$ then
26.     Send Resign(w(z(v))) to neighbours.
27. end if

SetRoot(u,r): // Update the information from neighbour u
 1. $r(u) \leftarrow r$
 2. if $(p(v) = u) \wedge (r(v) \neq r)$ then
 3.     $r(v) \leftarrow r$
 4.     Send SetRoot(v,r(v)) to neighbours.
 5. end if

Resign(w): // Resign from the clusterhead role
 1. if $(p(v) = v) \wedge (w(v) \leq w)$ then
 2.     Update $N_2(v), y(v)$.
 3.     if $N_2(v) \neq \emptyset$ then
 4.         $p(v) \leftarrow y(v)$
 5.         if $(p(v) = v)$ then
 6.             $r(v) \leftarrow p(v)$
 7.         else
 8.             $r(v) \leftarrow r(p(v))$
 9.         end if
10.         Send SetRoot(v,r(v)) to neighbours.
11.     end if
12. end if
```

  - $N_2(v) = \{u \in N_1(v) \mid r(u) = u\}$
  - $P_1(v) : ((r(v) = \perp) \vee (r(v) \neq v \wedge r(v) \notin N_2(v)) \vee (r(v) = v \wedge w(v) < w(y(v)))$
  - $P_2(v) : (\{u \in N_1(v) \mid r(u) \neq \perp\} = \emptyset)$
  - $P_3(v) : false$

  Tandem is an algorithm that considers semantic relationships among pairs of nodes as the main clustering criteria. Therefore, the set $N_1(v)$ is restricted to comprise only the nodes which are semantically similar, indicated by $s(v, u)$.

  Similar to DMAC, by limiting the set $N_2(v)$ to include only root nodes, Tandem generates one-hop clusters.

  Changing the parent (condition $P_1(v)$) is triggered only if the node is unassigned ($r(v) = \perp$), the root is no longer in set $N_2(v)$ ($r(v) \neq v \wedge r(v) \notin N_2(v)$) or node v is clusterhead and it has not the highest weight in $N_2(v)$ ($r(v) = v \wedge w(v) < w(y(v))$).

  The condition on which a node can become clusterhead ($P_2(v)$) depends on the set of nodes that are already in $N_1(v)$ and are assigned. If this set is not empty, Tandem prevents node v from becoming clusterhead, in order to minimize the effect of the erroneously perceived semantic similarity between neighbouring nodes. Otherwise, node v elects itself as clusterhead.

- **G-DMAC**: G-DMAC differs from the other algorithms by defining the following set of constraints for the clustering structure:

  - $\boldsymbol{h}$: an assigned node v can change its current parent $p(v)$ only if the new parent p_1 has a significant higher weight, i.e. $w(p_1) - w(p(v)) > h$, where h represents the minimum difference between the weights of p_1 and $p(v)$.
  - $\boldsymbol{k}$: if v is a root node, the number of root nodes that are allowed to be present in the neighbourhood of v is at most k; formally, $|\{v \in \Gamma^+(v) \mid p(v) = v\}| \leq k$; parameter k is used by condition $P_3(v)$ to determine whether any of the neighbours of v has to resign.

  To control the number of clusterheads that are allowed to be neighbours, G-DMAC uses a *Resign* message with parameter $w(z(v))$, which is the weight of the first clusterhead that violates the k-neighbourhood condition.

Neighbour $z(v)$ is defined such that $z(v) \in \{u \in N_1(v) \mid r(u) = u\}$ and $|\{u \in N_1(v) \mid r(u) = u \wedge w(u) > w(z(v))\}| = k$.

Using the additional input h and k, G-DMAC is defined by the following instantiations of $N_1(v), N_2(v), P_1(v), P_2(v), P_3(v)$:

– $N_1(v) = \Gamma^+(v)$

– $N_2(v) = \{u \in N_1(v) \mid w(u) > w(v) \wedge r(u) = u\}$

– $P_1(v) : ((r(v) = \perp) \vee (w(r(v)) + h < w(y(v))) \vee ((r(v) \neq v) \wedge (r(v) \notin N_2(v))) \vee ((r(v) = v) \wedge (|\{u \in N_1(v) \mid r(u) = u\}| > k) \wedge (w(v) \leq min\{w(z) \mid z \in \{u \in N_1(v) \mid r(u) = u\}\})))$

– $P_2(v) : true$

– $P_3(v) : ((r(v) = v) \wedge (|\{u \in N_1(v) \mid r(u) = u\}| > k) \wedge (w(v) > w(z(v))))$

Similar to DMAC and Tandem, by limiting the set $N_2(v)$ to include only root nodes, G-DMAC constructs only one-hop clusters.

Changing the parent (condition $P_1(v)$) occurs if the node is unassigned ($r(v) = \perp$), the root is no longer in v's neighbourhood ($(r(v) \neq v) \wedge (r(v) \notin N_2(v))$) the weight of the new clusterhead exceeds the current weight of the root with a certain threshold h ($w(r(v)) + h < w(y(v))$) or the number of roots exceeds k, v is root and has the lowest weight among the roots in its neighbourhood ($(r(v) = v) \wedge (|\{u \in N_1(v) \mid r(u) = u\}| > k) \wedge (w(v) \leq min\{w(z) \mid z \in \{u \in N_1(v) \mid r(u) = u\}\})$).

If the number of allowed root nodes exceeds the threshold k (condition $P_3(v)$), a *Resign* message will be sent to the neighbours to correct this situation.

In the following, we comment on a particular aspect that differentiates C4SD from DMAC and G-DMAC. As explained earlier, C4SD does not impose a limit on the maximum cluster height. Therefore, in principle, the algorithm can yield clusters with arbitrary height. In the worst case, a cluster can have a height of $|V|$. The advantage is that each node in C4SD chooses a parent independently of the decisions of its neighbours. On the contrary, DMAC and G-DMAC impose a maximum cluster height of one. This feature is useful to minimize the communication overhead with the clusterhead and to prevent the construction of large, unmanageable clusters. However, to achieve a maximum cluster height of one, each node v chooses as parent a neighbour which is already

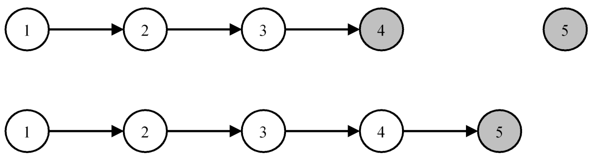


Figure 4.1: C4SD worst case, cluster height=$|V|$.

a root, and therefore its decision is dependent on the decisions of its neighbours. At the moment the chosen parent is no longer a root, v has to find another parent or to become itself a root. This phenomenon can lead to a *chain reaction* that may trigger reclustering in the whole network [45]. Note that although Tandem also imposes a maximum cluster height of one, it does not suffer from the chain reaction problem because (1) groups of nodes that are semantically separated do not influence each other in the decision process, and (2) a node with the highest weight in its neighbourhood does not become root (and thus does not attract other assigned nodes in its cluster) unless all its neighbours from N_1 are unassigned.

Figures 4.1 and 4.2 exemplify the worst case scenario for C4SD and the chain reaction for DMAC. The root nodes are indicated with gray filled circles, the dotted lines connect two neighbours and the arrows denote the parent-child relationship. The numbers represent the weights associated with nodes. Figure 4.1 shows the worst case for C4SD: linear topology and cluster height equal to $|V|$. When node 5 enters the network, each node keeps its current parent, except node 4, which chooses node 5 as parent. The clusterhead thus changes to node 5, but for the rest, the original structure remains. As an observation, we show in Chapter 5 that although theoretically C4SD allows clusters of arbitrary height, in practice the average cluster height is lower than 2, and in 95% of the cases it is below or equal to 3. This result shows that in practice, C4SD leads to acceptable clustering solutions for dynamic WSNs.

Figure 4.2 shows the same topology, but DMAC is used for clustering. The network is initially disconnected, with node 5 being isolated and forming a cluster with one member. Two other clusters are created with clusterheads 2 and 4, having height equal to one. When node 5 joins the network, a chain reaction is triggered: node 4 gives up its role as a clusterhead and joins node 5. Node 3 becomes a clusterhead and attracts node 2 as a member. Node 1 becomes the root of a cluster with only one node.

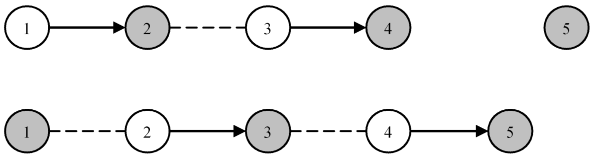


Figure 4.2: DMAC chain reaction.

Figures 4.1 and 4.2 illustrate that small differences in the definition of the sets $N_1(v)$ and $N_2(v)$ and conditions $P_1(v), P_2(v), P_3(v)$ may lead to completely different clustering structures, making thus Algorithm 1 a general framework for the class of weight-based clustering algorithms designed for mobile networks.

4.2 Correctness of the cluster formation

In the following, we prove formally that the considered algorithms fulfil the properties described in Section 4.1.2. Properties 1 and 2 are general properties, while Property 3 is applicable only if the network stabilizes. We prove that if certain *constraints* regarding the algorithm operation are satisfied, Algorithm 1 fulfils Properties 1-3. Thus, any algorithm that follows Algorithm 1 and satisfies the constraints is correct, in the sense that it fulfils Properties 1-3. Section 4.3 shows that each of the four algorithms, C4SD, DMAC, G-DMAC and Tandem does satisfy the constraints, and thus each algorithm fulfils Properties 1-3.

4.2.1 General properties

In this section, we prove Properties 1 and 2, which hold also for dynamic networks. First, we prove that Property 1 is always achieved by Algorithm 1. Next, we present a constraint (Constraint 1), which ensures that Property 2 is fulfilled.

4.2.1.1 Disjoint clusters

Lemma 1 *If a clustering algorithm follows the generalized algorithm, the network contains only disjoint clusters.*

Proof From the description of Algorithm 1, we deduce that every assigned node has only one parent, corresponding to the variable $p(v)$. Since a node belongs to the cluster of its parent, no node can be part of different clusters. □

4.2.1.2 No cycles

Constraint 1 *For every node v, at least one of the following propositions is true:*

1. *Node v may select as parent only itself or another node with a higher weight. Formally, $N_2(v) \subseteq \{x \in N_1(v) \mid w(x) > w(v)\}$.*
2. *Node v may select as parent only itself or a root node. If v is a root node that chooses another parent, it is allowed to select as parent only a root with higher weight. Formally, $(r(v) = v \wedge P_1(v) = true \wedge N_2(v) \neq \emptyset) \Longrightarrow (r(y(v)) = y(v) \wedge w(v) < w(y(v)))$*

We denote with $p_k(v)$ the parent of order k, i.e. $p_1(v) = p(v)$, $p_k(v) = p(p_{k-1}(v))$ for $k > 1$.

Lemma 2 *If a clustering algorithm follows the generalized algorithm and satisfies Constraint 1, the clustering structure does not contain cycles. Formally, $\forall v \in V$, if $p(v) \neq v$ then $p_k(v) \neq v, \forall k > 1$.*

Proof If the first proposition of Constraint 1 is satisfied, then each node picks exactly one parent, either itself or a neighbour of higher weight. Therefore, the network does not contain cycles.

If the second proposition of Constraint 1 is satisfied, then we have only one-hop clusters. If a node that is not root selects as parent another node that is a root, then v is added to the cluster of that root with a 1-hop distance. The root node cannot choose node v as parent, since node v is not a root. If two roots r_1 and r_2 become connected and both might choose the other as parent (i.e. $r_1 \in N_2(r_2)$ and $r_2 \in N_2(r_1)$), then at least the one with the highest weight remains root. Therefore, the network does not contain cycles. Note that if the two nodes get connected (i.e. the root with smaller weight selects the other root as parent), transitory two-hop clusters may occur. However, this situation is corrected in the next round, since now the 2-hop nodes do not have a root as parent and thus they must select a new parent. □

4.2.2 Stabilization property

In this section, we present two more constraints and show that these constraints imply Property 3.

We assume that starting from an arbitrary global state, the topological structure and the semantic similarities among nodes remain constant (i.e. $N_1(v)$ remains constant, $\forall v \in V$). We denote this moment with round 0.

We refer to round k, $k \geq 1$, as the series of consecutive rounds following 0. We denote with S^k the set of stabilized nodes corresponding to round k, i.e. $S^k = \{v \in V \mid \forall j \geq k,\ p^j(v) = p^k(v),\ r^j(v) = r^k(v)\}$, where $p^i(v)$ and $r^i(v)$ are the parent and root of v at round i.

We say that the clustering structure is *stable* if for any subsequent round, every node keeps the same parent and root. Formally, the structure is stable or reaches stability at round k if $S^k = V$.

4.2.2.1 Constraints

After round 0, we assume that the generalized algorithm satisfies the following constraints (see Section 4.3, where the constraints are proven for each special case of Algorithm 1) :

Constraint 2 *There exists $d \leq |V|$ such that after round 0 every d rounds either there exists at least one additional node that remains with the same parent and root in any subsequent round, or the network gets stable. Formally, $\exists d \leq |V|$, such that we have either $(\exists x \in S^{k+d}$ with $x \notin S^k)$ or $(S^{k+d} = V)$, $\forall k \geq 0$.*

Constraint 3 *Let $v \in V$. After the first round, all the nodes that are eligible to become parents of v are assigned. Formally, $\forall x \in N_2(v),\ p^1(x) \neq \perp$.*

4.2.2.2 Correctness proofs

Lemma 3 *Given Constraint 2, after at most $O(d|V|)$ rounds, the network stabilizes ($\exists k,\ 0 \leq k \leq d|V|$, such that $S^k = V$).*

Proof Constraint 2 implies that $|S^{k+d}| \geq |S^k| + 1$ if $|S^k| < V$ and $k \geq 0$. Therefore, we have that $|S^{d|V|}| = |V|$, i.e. after at most $O(d|V|)$ rounds the network stabilizes. $\square$

Lemma 4 *Given Constraint 3, a stabilized network does not have any unassigned parent node. Formally, let $x \in V$ be a node in the network. If $\exists v \in V$ such that $p(v) = x$, then $p(x) \neq \perp$.*

Proof From Constraint 3 we have that after one round, for any node $v \in V$, all the nodes in the set $N_2(v)$ from where v can choose a parent are assigned $(\forall x \in N_2(v),\ p(x) \neq \perp)$. □

4.2.3 Correctness of the generalized algorithm

Theorem 1 *Any algorithm which follows Algorithm 1 and satisfies Constraints 1-3 stabilizes after a finite number of rounds to a feasible state.*

Proof If Constraint 2 holds, then from Lemma 3 we have that after at most $O(d|V|)$, the network stabilizes.

From the definition of feasibility from Section 4.1.2, we deduce that the clustering structure is in a feasible state if: (1) the structure does not contain cycles, (2) the root of node v is also the root of $p(v)$, (3) there are no unassigned parents. We prove each of these properties in turn:

1. If Constraint 1 holds, then from Lemma 2 we have that the structure does not contain cycles.

2. The *SetRoot* message is propagated to all the nodes in the cluster whenever there is a topological or root change. Therefore, each node v learns the root of its cluster from $p(v)$, such that the root of v becomes equal to the root of its parent (see line 3 from the *SetRoot(u,r)* event, Algorithm 1).

3. If Constraints 3 holds, then from Lemma 4 we have that a stabilized network does not have any unassigned parent node.

Therefore, we have that after a finite number of rounds the network stabilises to a feasible state. □

4.3 Proofs of constraints for each algorithm

From Section 4.2.3, we have that any algorithm which follows Algorithm 1 and satisfies Constraints 1-3 is correct. Therefore, we prove these constraints for the four special cases of Algorithm 1 presented in Section 4.1.5, from where it follows that these algorithms are correct.

4.3.1 C4SD

Constraint 1 Since $N_2(v) = \{u \in N_1(v) \mid w(u) > w(v)\}$, the first proposition is always true. $\square$

Constraint 2 We may take $d = 1$. We first prove the constraint for the first round, and then we generalize for round $k > 1$.

First, note that $\forall v \in V$, $N_2(v)$, $P_1(v)$ and $P_2(v)$ do not change after round 1. Therefore, after the first round, each node has a stable parent. Let v^1 be the node with the maximum weight, $w(v^1) = max\ \{w(v) \mid v \in V\}$. Because $N_2(v^1) = \emptyset$ and $P_1(v^1) = true$, after the first round, v^1 becomes root. Any subsequent round does not change $N_2(v^1)$ and $P_1(v^1)$, and thus v^1 remains root.

At round k, $k > 1$, let v^k be the node with the highest weight that is not already stabilized (does not know its root $r(v)$), i.e. $w(v^k) = max\{w(v) \mid v \in V \setminus S^{k-1}\}$. Since the parent of v^k has a higher weight than v^k, we have that $p(v^k) \in S^{k-1}$, and therefore, it knows its root. Thus, in round k, also node v^k gets to know its root and stabilizes.

$\square$

Constraint 3 After the first round, all the nodes are assigned ($P_1(v) = true$, $P_2(v) = true$, $\forall v \in V$). Therefore, after the first round, there is no unassigned parent. $\square$

4.3.2 DMAC

Constraint 1 Since $N_2(v) = \{u \in N_1(v) \mid w(u) > w(v) \wedge r(u) = u\}$, the first proposition is always true. $\square$

Constraint 2 We may take $d = 1$. We first prove the constraint for the first round, and then we generalize for round $k > 1$.

Let v^1 be the node with the maximum weight, $w(v^1) = max\ \{w(v) \mid v \in V\}$. Because $N_2(v^1) = \emptyset$ and $P_1(v^1) = true$, after the first round, v^1 becomes root. Any subsequent round does not change $N_2(v^1)$ and $P_1(v^1)$, and thus v^1 remains root.

At round k, $k > 1$, let v^k be the node with the highest weight that is not already stabilized, $w(v^k) = max\{w(v) \mid v \in V \setminus S^{k-1}\}$. The decision of node v^k depends only on the decisions of the neighbours with higher weights, which are already stabilized. Therefore, the set $N_2(v^k)$ does not change in any subsequent round, as well as $P_1(v^k)$ and $P_2(v^k)$, so that at round k node v^k stabilizes, by selecting a parent that is also a root. $\square$

Constraint 3 The set $N_2(v)$ contains only assigned nodes, $\forall v \in V(N_2(v) = \{u \in N_1(v) \mid w(u) > w(v) \wedge r(u) = u\})$. $\square$

4.3.3 G-DMAC

Constraint 1 Since $N_2(v) = \{u \in N_1(v) \mid w(u) > w(v) \wedge r(u) = u\}$, the first proposition is always true. $\square$

Constraint 2 We may take $d = 1$. We first prove the constraint for the first round, and then we generalize for round $k > 1$.

Let v^1 be the node with the maximum weight, $w(v^1) = max\ \{w(v) \mid v \in V\}$. Node (v^1) is either unassigned, or is root. If v^1 is unassigned, since $N_2(v^1) = \emptyset$ and $P_2(v) = true$, v^1 becomes root. Otherwise, v^1 remains root in any subsequent round.

At round k, $k > 1$, let v^k be the node with the highest weight that is not already stabilized, i.e. $w(v^k) = max\{w(v) \mid v \in V \setminus S^{k-1}\}$. The decision of node v^k depends only on the decisions of the neighbours with higher weights, which are already stabilized. Therefore, the set $N_2(v^k)$ does not change in any subsequent round, as well as $P_1(v^k)$ and $P_2(v^k)$.

A *Resign* message received by v^k, which can trigger the resignation of v^k from being a root, can be sent only by a node with a higher weight than v^k. Therefore, if the k-neighbourhood condition is violated for any of the stabilized neighbouring nodes of v that are also roots, v may receive a *Resign* message and act accordingly (i.e. if v is a root, it gives up its role and selects a new parent) only in a round preceding round k. Therefore, node v selects a parent that is also a root at round k and stabilizes. $\square$

Constraint 3 The set $N_2(v)$ contains only assigned nodes, $\forall v \in V(N_2(v) = \{u \in N_1(v) \mid w(u) > w(v) \wedge r(u) = u\})$. $\square$

4.3.4 Tandem

Constraint 1 We prove the second proposition. A node can choose as parent only itself or a root node, because the set $N_2(v)$ contains only root nodes $(N_v(v) = \{u \in N_1(v) \mid r(u) = u\})$. Since $y(v) \in N_2(v)$, we have that

$r(y(v)) = y(v)$. If node v is a root, it can give up its role by selecting as parent only a root node of higher weight. From the definition of $P_1(v)$ ($P_1(v)$: $((r(v) = \perp) \vee (r(v) \neq v \wedge r(v) \notin N_2(v)) \vee (r(v) = v \wedge w(v) < w(y(v)))))$, we have that $(r(v) \neq v) \vee (r(v) = v \wedge w(v) < w(y(v)))$. $\square$

Constraint 2 All the nodes that have $N_1(v) = \emptyset$ become and remain unassigned after the first round. In the following, we discuss the nodes that have $N_1(v) \neq \emptyset$.

We may take $d = 2$. We first prove the constraint for the first 2 rounds, and then we generalize for round $k > 2$.

Suppose that after round 0, none of the nodes is a root. If condition P_2 is satisfied by at least one node v ($\exists v \in V$ such that $\{u \in N_1(v) \mid r(u) \neq \perp\} = \emptyset$), then after the first round v becomes a root. Otherwise, all the nodes in V become unassigned after the first round ($p(v) = \perp$, $\forall v \in V$). In this case, after the second round, the condition P_2 becomes satisfied by least one node, who becomes root. Let v^1 be the root node with the highest weight after the second round, $r(v^1) = v^1$ and $w(v^1) = max\{w(v) \mid r(v) = v\}$. Node v^1 remains root in any subsequent round because none of its neighbours can become root ($N_2(v) \neq \emptyset$, $\forall v \in N_1(v^1)$, by Assumptions 3 and 4), so it stabilizes at round 2.

Starting from the stabilized set at round k, we prove that after at most 2 rounds, the stabilized set contains at least one additional node.

At round $k+1$, $k \geq 2$, if exists a node x that has as parent a node from the already stabilized set ($x \notin S^k$, $p(x) \in S^k$), we take $v^{k+1} = x$. Due to condition $P_1(v^{k+1})$, node v^{k+1} does not change the root in any subsequent round and thus v^{k+1} stabilizes at round $k+1$.

If none of the nodes $x \notin S^k$ has as parent a node from the already stabilized set, we take v^{k+1} the root node with the maximum weight which is not part of the stabilized set ($r(v^{k+1}) = v^{k+1}$ and $w(v^{k+1}) = max\{w(v) \mid r(v) = v,\ v \in V \setminus S^k\}$). Node v^{k+1} does not change the role of root in any subsequent round because none of its neighbours can become root ($N_2(v) \neq \emptyset$, $\forall v \in N_1(v^{k+1})$), so it stabilizes at round $k+1$.

The last case is when all the nodes which are not yet stabilized are unassigned, i.e. $\forall x \notin S^k$, $r(x) = \perp$. If all these nodes have assigned neighbours, then from condition $P_2(x)$ it follows that all these nodes remain unassigned in any subsequent round, so the network stabilizes. Otherwise, there exists at least one node that has all neighbours unassigned, i.e. $\exists x \notin S^k$ such that $P_2(x)$ is true $((\{u \in N_1(x) \mid r(u) \neq \perp\} = \emptyset)$. Therefore, node x becomes root at round $k+2$. At round $k+2$, we take v^{k+2} the root node with the maximum weight which is not part of the stabilized set. Node v^{k+2} does not change the role of root in any subsequent round because none of its neighbours can become root ($N_2(v) \neq \emptyset,\ \forall v \in N_1(v^{k+2})$), so it stabilizes at round $k + 2$. $\square$

Constraint 3 The set $N_2(v)$ contains only assigned nodes, $\forall v \in V$ ($N_2(v) = \{u \in N_1(v) \mid r(u) = u\}$). $\square$

4.4 Conclusions

This chapter presents a generalized clustering algorithm that makes localized decisions based on 1-hop neighbourhood information and produces disjoint clusters. The algorithm represents a generalization of weight-based clustering algorithms designed for mobile ad-hoc and sensor networks, by using a set of general variables and conditions. Concrete algorithms, leading to different clustering structures, can be defined by giving specific meaning to these variables and conditions. As examples, we present four specialized algorithms, namely C4SD (see Chapter 5), Tandem (see Chapter 7), DMAC [43] and G-DMAC [42]. Assuming as valid a set of given constraints, this generalization allows us to define and prove common properties for these algorithms. Any new algorithm that follows the generalized clustering algorithm can be proven correct, with respect to the properties given in Section 4.1.3, by proving that it satisfies this set of constraints.

Appendix

Summary of notation:

Notation	**Explanation**
$\Gamma(v)$	The open neighbourhood of v, i.e. $\Gamma(v) = \{u \in V \mid (u, v) \in E\}$
$\Gamma^+(v)$	The closed neighbourhood of v, i.e. $\Gamma^+(v) = \Gamma(v) \cup \{v\}$
$w(v)$	The weight of v
$s(v, u)$	The semantic relationship between v and u
$\perp$	The symbol for unassigned nodes
$p(v)$	The parent of v, $p(v) \in V \cup \{\perp\}$
$r(v)$	The root or clusterhead of v, $r(v) \in V \cup \{\perp\}$
$N_1(v)$	The subset of neighbours with which node v may cluster, $N_1(v) \subseteq \Gamma^+(v)$
$N_2(v)$	The subset of neighbours with which node v may cluster, which can become parents of v, $N_2(v) \subseteq N_1(v)$
$P_1(v)$	The condition on which v chooses a different parent
$P_2(v)$	The condition on which v becomes root
$P_3(v)$	The condition on which v informs the neighbours that one of them has to resign from the clusterhead role
$y(v)$	The neighbour of v from $N_2(v)$ with the highest weight, i.e. $y(v) \in N_2(v)$, $w(y(v)) = max\{w(u) \mid u \in N_2(v)\}$
$z(v)$	The neighbour of v from $N_1(v)$ with the highest weight that has to resign from the clusterhead role, based on condition $P_3(v)$

Table 4.1: Summary of notation.

Chapter 5

Cluster-based service discovery for heterogeneous wireless sensor networks

We propose an energy-efficient service discovery protocol for heterogeneous wireless sensor networks. Our solution exploits a cluster overlay, where the clusterhead nodes form a distributed service registry. A service lookup results in visiting only the clusterhead nodes. We aim for minimizing the communication costs during discovery of services and maintenance of a functional distributed service registry. To achieve these objectives, we propose a clustering algorithm which (1) makes decisions based on one-hop neighbourhood information, (2) avoids chain reactions, and (3) constructs a set of sparsely distributed clusterheads. We analyse how the properties of the clustering structure influence the performance of the service discovery protocol, by comparing theoretically and through simulations our proposed clustering algorithm with DMAC. To validate our simulations we implement the proposed solution on resource-constraint sensor nodes, and we measure the performance of the protocol running on different testbeds.

5.1 Introduction

The problem that we address in this chapter is the design of a service discovery protocol suitable for heterogeneous wireless sensor networks that reduces the workload of the resource-constraint devices and avoids the significant traffic induced by the traditional flood-based solutions in dense networks. We propose a solution based on clustering, where a set of nodes, selected based on their capabilities, acts as a distributed directory of service registrations for the nodes in their cluster. In this way, (1) the communication costs are reduced, since the service discovery messages are exchanged only among the directory nodes, and (2) the distribution of workload takes into account the capabilities of the nodes.

The main contributions of this chapter are therefore:

- C4SD (Clustering for Service Discovery), a lightweight clustering algorithm that builds a distributed directory of service registrations. By making decisions based only on the 1-hop neighbourhood information, the clustering algorithm reacts rapidly to topological changes.

- SD4WSN (Service Discovery for Wireless Sensor Networks), an energy-efficient service discovery protocol that exploits the underlying clustering structure.

Additionally, we compare C4SD with DMAC (Distributed Mobility-Adaptive Clustering) [43], a state of the art counterpart. DMAC constructs and maintains an independent dominating set, being designed for mobile networks. Nodes decide based on their one-hop neighbourhood information, which assures rapid reaction to topology changes. However, DMAC suffers from the *chain reaction* phenomenon, where a single topology change in the network may trigger significant changes in the dominating set. For a distributed directory composed of nodes from the dominating set, the chain reaction causes additional overhead for maintaining consistent service registries. We compare the performance of our proposed clustering algorithm with DMAC, when using them as structural basis for our service discovery protocol.

The remaining part of the chapter is organized as follows: we present in detail the clustering algorithm and the service discovery protocol for heterogeneous WSNs in Sections 5.2 and 5.3. We evaluate the performance of the proposed algorithms in Section 5.4. The implementation results are presented in Section 5.5. Section 5.6 presents the conclusions.

5.2 Clustering algorithm

In this section we present the design considerations, the network model and the detailed description of our clustering algorithm.

5.2.1 Design considerations

C4SD constructs an overlay network, which facilitates the discovery of services in an energy-efficient fashion. We discuss from the design perspective several techniques for reducing the communication cost during (1) discovery of services and (2) maintenance of the distributed directory.

During the discovery process, messages are exchanged among the clusterhead nodes. To minimize the discovery cost, the root nodes have to be sparsely distributed on the deployment area, as a high density of root nodes (or clusters) would lead to a high communication cost during discovery. Therefore, the clustering algorithm should construct an *independent set* of clusterheads, i.e. two root nodes are not allowed to be neighbours.

In the following, we give the design considerations for minimizing the communication cost during the maintenance of the distributed directory:

- *Make decisions based on 1-hop neighbourhood information.* Clustering algorithms that require each node to have complete topology knowledge over a number of hops are expensive with regard to the maintenance cost. These algorithms generally require that nodes are static during the initial cluster setup. We aim to build a lightweight clustering structure that requires only the 1-hop neighbourhood topology information, and does not impose an initial stationary topology.

- *Avoid chain reactions.* A chain reaction occurs when a single network topology change leads to reclustering throughout the network. For a distributed directory composed of clusterhead nodes, a chain reaction leads to high overhead for maintaining consistent service registries. Therefore, an energy-efficient solution should avoid chain reactions, such that local topology changes determine only local modifications of the directory structure.

- *Distribute the overhead depending on the capabilities of the nodes.* To minimize the maintenance effort and to relieve the resource-lean devices, the knowledge on adjacent clusters and the intra-cluster information should be distributed depending on the capabilities of the nodes.

In theory, building an independent set of root nodes and avoiding a chain reaction comes at the expense of constructing clusters with an arbitrary height (see Chapter 4). However, in practice, we can achieve small-height clusters without imposing a maximal height limit (see Section 5.4.2.2).

5.2.2 Network model

We use the network model defined in Chapter 4. Each node is assigned (1) a unique hardware identifier, termed the *address* of the node, and (2) a weight, termed the *capability grade*, representing an estimate of the node's dynamics and available resources. The higher the capability grade, the more suitable is the node for the clusterhead role. We introduce the term *capability grade* as a synonym for the *weight* measure from Chapter 4 to emphasise the fact that it is an estimation of the available resources present on the node. The capability grade can be calculated in accordance with the unified framework for clustering proposed by Nocetti et al. [133], as a weighted sum of the memory available on the node, the energy left, the processing power and the degree of dynamics (speed or just an indication of the typical state of the node, e.g. moving/stationary). We say that two trees are *adjacent* if there are two nodes, one from each tree, which are connected through a link.

Given a node v, we use the following notation, in addition to the general notation defined in Section 4.1:

- $c(v)$ is the capability grade of v, corresponding to the weight $w(v)$.
- $\Delta(v)$ is the set of children of node v, $\Delta(v) = \{u \in V \mid p(u) = v\}$
- $p_k(v)$ is the parent of order k, defined as $p_0(v) = v$, $p_{k+1}(v) = p(p_k(v))$
- $C(v)$ is the set of nodes that are part of the same cluster as v, $C(v) = \{u \in V \mid r(u) = r(v)\}$
- $T(v)$ is the set of nodes from the sub-tree rooted at v, $T(v) = \{u \in V \mid \exists k$ such that $p_k(u) = v\}$
- $R_u(v)$ is the set of adjacent clusters of node v, represented by their roots, which can be reached through node u, where $u \in \Gamma(v)$, i.e. $R_u(v)$ is either the root of u, if u is in a different cluster, or the roots of adjacent clusters reached through the children of u; $R_u(v)$ is described according to the following cases:

  - if $u \in \Delta(v)$, then $R_u(v)$ is the set of root nodes of clusters adjacent to the sub-tree rooted at v. Formally, $R_u(v) = \{r \in V \setminus \{r(v)\} \mid \exists x \in T(u), \exists y \in \Gamma(x) \text{ such that } r = r(y)\}$;
  - if $u \in C(v) \setminus \Delta(v)$, then $R_u(v) = \emptyset$;
  - if $u \notin C(v)$, then $R_u(v) = \{r(u)\}$.

- $S(v)$ is the set of services provided by node v
- $S_u(v)$ is the set of services registered to v by $u \in \Delta(v)$. Formally, $\forall u \in \Delta(v),\ S_u(v) = \{S(x) \mid x \in T(u)\}$.

5.2.3 Construction of clusters

The construction of clusters follows the generalized algorithm described in Chapter 4, where nodes choose a neighbour with higher capability grade as *parent*, while other nodes that do not have such a neighbour are *roots*. Algorithm 2 is a *special case* of Algorithm 1, where for each node v, the generic sets $N_1(v), N_2(v)$ and conditions $P_1(v), P_2(v), P_3(v)$ are replaced with the specific instances of C4SD. Algorithm 2 is also an *extended* version of Algorithm 1, by defining a new message *UpdateInfo* and maintaining the knowledge on adjacent clusters (see Section 5.2.4).

Briefly, the protocol works as follows:

- Nodes that have the highest capability grades among their neighbours declare themselves clusterheads and broadcast a *SetRoot* message announcing their roles.
- The remaining nodes choose as parent the neighbour with the highest capability grade.
- When a node receives a *SetRoot* message from its parent, it learns the cluster membership and rebroadcasts the *SetRoot* message.

5.2.4 Knowledge on adjacent clusters

The knowledge on adjacent clusters is constructed and held in the $R_u(v)$ sets using the *UpdateInfo* message. The *SelectParent* and SetRoot events from Algorithm 1 are extended to accommodate the construction and maintenance of

Algorithm 2: Clustering algorithm - node v (events/actions)

```
*Initialization*: // Parent is chosen
 1: $r(v) \leftarrow \perp$; $p(v) \leftarrow \perp$; $R_m(v) \leftarrow \emptyset,\ \forall m \in \Gamma(v)$

*SelectParent*: // Build the clustering structure by selecting the parent
 1: $r_0 \leftarrow r(v)$; $p_0 \leftarrow p(v)$; $\Gamma_0(v) \leftarrow \Gamma(v)$; $R_0 = \cup_{m\in\Gamma(v)} R_m(v)$
 2: Update $\Gamma(v),\ \Gamma^+(v),\ N_2(v),\ y(v)$      // $N_2(v) \leftarrow \{u \in \Gamma(v) \mid w(u) > w(v)\}$
 3: if $N_2(v) \neq \emptyset$ then
 4:     $p(v) \leftarrow y(v)$            // $y(v) \in N_2(v),\ c(y(v)) = max\{c(u) \mid u \in N_2(v)\}$
 5:     $r(v) \leftarrow r(p(v))$
 6: else if $(p(v) \neq v)$ then
 7:     $p(v) \leftarrow v$
 8:     $r(v) \leftarrow v$
 9: end if
10: if $(r(v) \neq r_0) \vee (\Gamma(v) \setminus \Gamma_0(v) \neq \emptyset)$ then
11:     Send *SetRoot(v,r(v))* to neighbours.
12: end if
13. $\forall m \in \Gamma(v),\ R_m(v) \leftarrow R_m(v) \setminus \{r(v)\}$
14. if $(p_0 \neq p(v)) \wedge (p_0 \in \Gamma(v))$ then
15.     Send *UpdateInfo* $(v, \emptyset)$ to $p(v)$
16. end if
17. if $(v \neq p(v)) \wedge ((p_0 \neq p(v)) \vee (R_0 \neq \cup_{m\in\Gamma(v)} R_m(v)))$ then
18.     Send *UpdateInfo* $(v, \cup_{m\in\Gamma(v)} R_m(v))$ to $p(v)$
19. end if

*SetRoot* $(u, r)$: // Receive root $r$ from neighbour $u$
 1. if $(p(v) = u) \wedge (r(v) \neq r)$ then
 2.     $r(v) \leftarrow r$
 3.     Send *SetRoot*$(v,\ r(v))$ to neighbours
 4. end if
 5. $R_0 = \cup_{m\in\Gamma(v)} R_m(v)$
 6. if $r(v) = r$ then
 7.     $\forall m \in \Gamma(v),\ R_m(v) \leftarrow R_m(v) \setminus \{r\}$
 8. else
 9.     $R_u(v) \leftarrow \{r\}$
10. end if
11. if $(v \neq p(v)) \wedge (R_0 \neq \cup_{m\in\Gamma(v)} R_m(v))$ then
12.     Send *UpdateInfo* $(v, \cup_{m\in\Gamma(v)} R_m(v))$ to $p(v)$
13. end if

*UpdateInfo* $(u, R)$: // Receive adjacent clusters $R$ from $u$
 1. $R_0 = \cup_{m\in\Gamma(v)} R_m(v)$;
 2. $R_u(v) \leftarrow R \setminus \{r(v)\}$
 3. if $(v \neq p(v))) \wedge (R_0 \neq \cup_{m\in\Gamma(v)} R_m(v))$ then
 4.     Send *UpdateInfo*$(v, \cup_{m\in\Gamma(v)} R_m(v))$ to $p(v)$
 5. end if
```

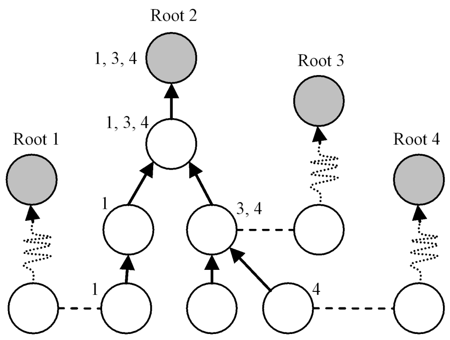


Figure 5.1: Nodes learn from neighbours which are the adjacent clusters (indicated by dashed lines) and propagate the knowledge to the parents (following the direction of arrows).

this knowledge. The extension starts at line 13 for the *SelectParent* event and at line 5 for the *SetRoot* event.

The extension works as follows: the root nodes learn about the adjacent clusters from the nodes present at the cluster borders. During the propagation of the broadcast message *SetRoot* down to the leaf nodes, the message is also received by nodes from adjacent clusters. These nodes store the adjacent root identity in their $R_u(v)$ sets and report it to their parents. The information is propagated up in the tree with a message which we term *UpdateInfo*. Through this message, nodes learn the next hops for the paths leading to the clusters adjacent to their sub-trees. In particular, the root nodes learn the adjacent clusters and the next hops on the paths to reach their clusterheads. Figure 5.1 gives an intuitive example of learning the adjacent clusters.

The events of receiving messages *SetRoot* and *UpdateInfo* from Algorithm 2 describe how the knowledge and the paths to adjacent clusters are updated for a given node v. Duplicate *UpdateInfo* messages are discarded: a node v sends the message *UpdateInfo* to its parent if and only if the set of known root nodes changes.

5.2.5 Maintenance in face of topology changes

We model the network dynamics by considering two events to which every node has to react: link addition and link deletion. These events are triggered by the lower layers of the communication stack. When a link is added or deleted to/from a node v, the *SelectParent* function is called, which adjusts the cluster membership accordingly (see Algorithm 2). The cluster membership changes in one of the following situations:

- A node discovers a new neighbour with a higher capability grade than its current parent. The node then selects that neighbour as its new parent.
- A node detects the failure of the link to its parent. The node then chooses as new parent the node with the highest capability grade in its neighbourhood.

Besides reclustering, topology changes may also require modifications in the knowledge on adjacent clusters. The *SetRoot* message informs nodes about the cluster membership of their neighbours, while the *UpdateInfo* message is used for transmitting updates from children to parents. We distinguish the following situations:

- A node v detects a new neighbour from a different cluster. Consequently, v adds the root of that cluster to its knowledge.
- A node v switches from parent p_0 to p_1. Then v notifies p_0 to remove the information associated with v and sends the list of adjacent clusters to p_1.
- A node v detects the failure of the link to one of its neighbours u. As a result, v erases the knowledge associated with u.
- Any change of global knowledge at node v results in transmitting the message *UpdateInfo* from v to its parent.

5.2.6 A clustering alternative: DMAC

We choose DMAC as a viable clustering alternative for our service discovery protocol. Its simplicity and good performance results [45] make it suitable for sensor environments. DMAC achieves fast convergence, as nodes decide their roles using only 1-hop neighbourhood information. The clusters are constructed based on unique weights assigned to nodes. The higher the weight, the more suitable the node is for the clusterhead role. The difference with our clustering algorithm is that DMAC imposes a maximum cluster height of one, whereas our protocol in principle may lead to an arbitrary cluster height (see Chapter 4). For the construction of clusters, DMAC uses two types of broadcast messages, *Clusterhead* and *Join*, announcing the roles of the nodes to their neighbours. The role decision of a node is dependent on the decisions of the neighbours with higher weights. Therefore, a single topology change may trigger reclustering of a whole chain of dependent nodes. This phenomenon is called a *chain reaction.* For a distributed directory composed of clusterhead nodes, the chain reaction leads to high overhead for maintaining consistent service registries. In Section 5.4.3 we study the impact of the cluster height and the chain reaction on the performance of the service discovery protocol, in comparison to our proposed clustering solution.

5.3 Service discovery protocol

During service discovery, service request messages look for a service match in the set of nodes which are part of the distributed directory. We first describe how the services are registered to the directory nodes and then we present the service discovery process.

5.3.1 Service registration

During service registration, each node sends the local service descriptions and the descriptions received from its children to the parent node. In this way, a node learns the service descriptions of the nodes placed lower in hierarchy in its cluster. In particular, a root node is informed of all the service descriptions offered by the nodes in its cluster. Since the registration process requires unicast messages to be transmitted from children to parents, it can be integrated with the transfer of knowledge on adjacent clusters. Thus, the communication cost is improved by using the same message *UpdateInfo* for both service registrations and transferring the knowledge on adjacent clusters. Algorithm 3 shows the integrated version of the *UpdateInfo* message, where a node updates the information on both the adjacent clusters and the known services.

In the following we describe how the distributed service registry is kept consistent when the topology changes. In the case of a parent reselection, a child node v registers the services from its sub-tree with the new parent p_1, and notifies the old parent p_0 (if it is still reachable) to purge the outdated service information. The process is transparent to the other nodes in the sub-tree rooted at v. If the overall service information at p_0 and p_1 changes due to the parent reselection, the modifications are propagated up in the hierarchy.

Algorithm 3: SD4WSN service registration - node v

```
   UpdateInfo $(u, R, S)$:
   // receive adjacent clusters $R$ and services $S$ from $u$
1.  $R_0 = \cup_{m \in \Gamma(v)} R_m(v)$
2.  $S_0 = \cup_{m \in \Delta(v)} S_m(v) \cup S(v)$
3.  if $R = \emptyset$ then
4.      $\Delta(v) \leftarrow \Delta(v) \setminus \{u\}$
5.      $S_u(v) \leftarrow \emptyset$
6.  else
7.      $\Delta(v) \leftarrow \Delta(v) \cup \{u\}$
8.      $S_u(v) \leftarrow S$
9.  end if
10. $R_u(v) \leftarrow R \setminus \{r(v)\}$
11. if $(v \neq p(v)) \wedge ((R_0 \neq \cup_{m \in \Gamma(v)} R_m(v)) \vee (S_0 \neq \cup_{m \in \Delta(v)} S_m(v) \cup S(v)))$ then
12.     Send $UpdateInfo(v, \cup_{m \in \Gamma(v)} R_m(v), \cup_{m \in \Delta(v)} S_m(v) \cup S(v))$ to $p(v)$
13. end if
```

5.3.2 Service discovery

SD4WSN uses a distributed directory of service registrations. Suppose a node in the network generates a service discovery request *ServDisc*. The request is first checked against the local registrations. In the case where no match is found, the message is forwarded to the parent. This process is repeated until the *ServDisc* message reaches the root of the cluster. When a root node receives a *ServDisc* message and it does not find a match in the local registry, the message is forwarded to the roots of the adjacent clusters. The next hop on the path leading to the adjacent cluster is decided by every node that acts as forwarder of the *ServDisc* message. Each node v along the path checks its $R_u(v)$ sets and picks a neighbour that has a path to the root of the adjacent cluster. In the case where a link is deleted and v cannot forward the *ServDisc* message, it chooses another neighbour that provides a path to destination. If such a neighbour does not exist, v informs its parent that it no longer has a route to the next cluster. The same procedure is repeated until all the paths to destination are tested. If the next cluster is not reachable, the root node erases the cluster from its knowledge.

The result of a service search is typically the address of one or more service providers. This response can be returned by the first node that finds a match in its registry for the requested service. However, in certain situations it may be preferable that the service provider itself issues a reply for the service request. Examples include applications where service descriptions change frequently, or cases where the reply incorporates more information than the address of the

Algorithm 4: SD4WSN service discovery - node v

```
    ServDisc $(u, s, d, f)$:
    // receive message ServDisc from node $u$, requesting service $s$, destination $d$, flag $f$
 1. if $f = TRUE$ then
 2.    if $s \in \cup_{m\in\Delta(v)} S_m(v) \cup S(v)$ then
 3.       Service found; generate reply
 4.    else if $p(v) = v$ then
 5.       for all $r \in \cup_{m\in\Gamma(v)} R_m(v)$ do
 6.          Pick $m \in \Gamma(v)$ such that $r \in R_m(v)$
 7.          Send $ServDisc(v, s, r, TRUE)$ to $m$
 8.       end for
 9.    else if $d = r(v)$ then
10.       Send $ServDisc(v, s, d, TRUE)$ to $p(v)$
11.    else if $d \in \cup_{m\in\Gamma(v)} R_m(v)$ then
12.       Pick $m \in \Gamma(v)$ such that $d \in R_m(v)$
13.       Send $ServDisc(v, s, d, TRUE)$ to $m$
14.    else
15.       Send $ServDisc(v, s, d, FALSE)$ to $p(v)$
16.    end if
17. else
18.    $R_u(v) \leftarrow R_u(v) \setminus \{d\}$
19.    if $d \in \cup_{m\in\Gamma(v)} R_m(v)$ then
20.       Pick $m \in \Gamma(v)$ such that $r \in R_m(v)$
21.       Send $ServDisc(v, s, d, TRUE)$ to $m$
22.    else if $p(v) \neq v$ then
23.       Send $ServDisc(v, s, d, FALSE)$ to $p(v)$
24.    end if
25. end if
```

node. In these situations, the *ServDisc* message is forwarded down the cluster until it reaches the service provider. In the case where the link to the service provider is deleted or the service description is no longer valid, the service request is sent back to the root node which forwards it to the adjacent clusters.

The service discovery reply may follow the reverse cluster-path to the client, or any other path if a routing protocol is available. For the first case, if there is a cluster partition, the path can be reconstructed using the same search strategy as for the *ServDisc* message, where this time the service is the address of the client.

Algorithm 4 describes SD4WSN, where replies are generated by nodes in the distributed directory. The message *ServDisc* has four parameters: the neighbour u that sends the request, the service description s, the final destination d of the message (typically a root node) and a flag f. The flag indicates whether the message is a fresh service discovery request, or it is a failure notification of a previous attempt to reach an adjacent cluster. In the latter case, the failed route is erased from the knowledge on adjacent clusters and another message is sent using an alternate path. To summarize, Algorithm 4 has the following properties:

Property 4 *If G is a connected graph representing a static network, all the root nodes are visited and thus the service is eventually found.*

Property 5 *If the network is dynamic, such that links are deleted or new links appear, alternate paths are built up and used during service discovery to compensate for topological changes.*

In the following, we argue that the above properties hold for the SD4WSN protocol. We describe the actions taken by a node v receiving a *ServDisc* message, depending on its role in the clustering structure and its acquired knowledge on adjacent clusters. If node v receives a *ServDisc* message with flag $f = TRUE$ and does not have a service registration for s, we have the following cases:

- v is root: for every adjacent cluster of v with root r, v picks a neighbour that has a path to r, and sends the *ServDisc* to this neighbour with the final destination r.
- The final destination of the *ServDisc* message is the root of v: v sends the message to its parent.
- The final destination of the *ServDisc* message is the root r of an adjacent cluster: v picks a neighbour that has a path to r, and sends the *ServDisc* to this neighbour.
- v is neither root, nor has a path to the destination: this is the case where a link was broken and thus v is not connected anymore to the destination cluster; v informs its parent of the loss, by sending the *ServDisc* message with flag $f \leftarrow FALSE$.

The first three cases ensure that Property 4 holds, because in a static connected network, if the service is not found in the local cluster, the *ServDisc* message visits all the root nodes in the network.

The last case applies to a dynamic network, where alternate paths are built up and used if the link to the destination brakes, directly referring to Property 5. When a node v receives a *ServDisc* message from one of its children, with flag $f = FALSE$ and destination r, v deletes the link to r through this child and

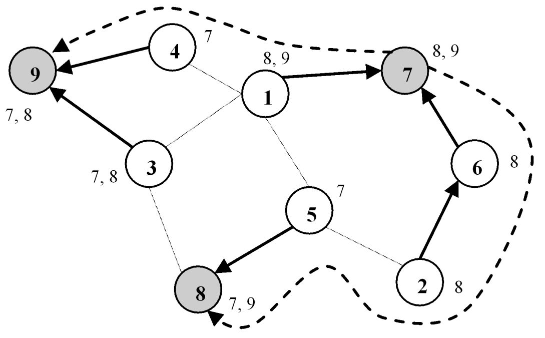


Figure 5.2: Example of a clustered network and the propagation of service discovery messages.

searches for another path to r. If v does not have another path to r, it transmits the message to its parent. In this way, the message may reach the root if there is no alternate path. If an alternate path is found, the *ServDisc* message is transmitted on this path with flag $f = TRUE$.

Let us take a look at the example from Figure 5.2. A network with nine nodes is partitioned in three clusters, following the C4SD algorithm. The root nodes are indicated with gray filled circles. The lines show the links between adjacent clusters, while the straight arrows indicate a parent-child relationship. The local knowledge on adjacent clusters is displayed next to each node. For example, node 2 has knowledge about the adjacent root 8. This information is aggregated at the higher levels in the hierarchy, such that the root nodes inherit the information acquired by all the nodes in their cluster. Root node 7 has therefore information about adjacent clusters 8 and 9. The dotted arrows show an example of how service discovery messages travel from root node 7 to the adjacent clusters 8 and 9, by using Algorithm 4.

For an improved behaviour in face of mobility, SD4WSN can be extended with *caching* techniques. This extension works as follows: a root node caches for a limited period of time the *ServDisc* messages for which the root node did not reply. If a newly arrived node registers a service for which there is a match in the cache, the root node can respond to the old service request. When a root node is notified on a new adjacent cluster, it sends the valid service request entries from its cache to the new clusterhead. As a result, the overall hit ratio is improved.

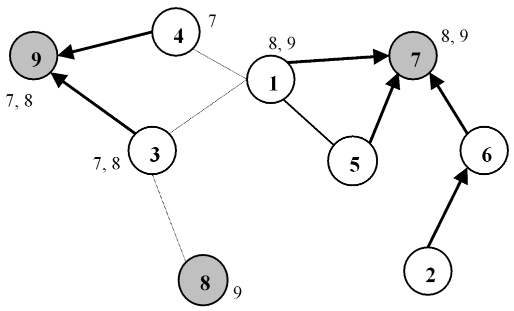


Figure 5.3: The clustered network from Figure 5.2, where node 5 has joined cluster 7.

Let us consider the example from Figure 5.2. Suppose that node 7 receives from node 6 a service discovery message, for a service provided by node 5. Node 7 does not have the service, so it forwards the message to the adjacent clusters 8 and 9. In the meantime, node 5 moves towards node 7 and it becomes the new child of node 7, as shown in Figure 5.3. Therefore, the service from node 5 is not provided any more by cluster 8, so the discovery fails. However, if node 7 caches the service discovery message for a limited amount of time, when node 5 joins the new cluster, node 7 can check whether the services provided by node 5 match any of the cached *ServDisc* messages. In this case, the matching is successful and the service is found.

5.4 Performance evaluation

We evaluate the performance of the combined solution through both a theoretical and a simulation-based analysis. By comparing the properties and simulating both C4SD and DMAC, we show the impact of the chosen clustering algorithm on the performance of SD4WSN.

5.4.1 Simulation settings

For our experiments we use the OMNeT++ [157] simulation environment. We generate a random network, by placing N nodes uniformly randomly distributed on a square area of size $a \times a$, as suggested by the motivating scenario of Chapter 1. We consider links to be bidirectional, so all nodes have the same transmission range, r. There is a link between two nodes if the distance between them is less or equal to r. We analyse the performance of our proposed algorithms under different network densities, by varying the transmission range and/or the number of nodes .

In a heterogeneous network, the capabilities of the nodes can be assumed uniformly randomly distributed. For example, the infrastructure of beacons used for localization in a WSN, which have the same level of capabilities, is modelled with either a uniform grid placement [50], or using a stochastic uniform distribution [77]. Applications that require sensing and control over a large area employ resource-rich nodes capable of acting, which are uniformly distributed on the entire region [30]. Both resource-constrained and powerful nodes in a heterogeneous sensor network are uniformly distributed over the implementation area for the purpose of surveillance [125]. Following this remark, the nodes in our simulations choose their capability grades from a uniform distribution. We also assure that static nodes have higher capability grades than mobile nodes.

We compare the performance of C4SD and DMAC under the same topological conditions. We extend DMAC with the method for maintaining the knowledge on adjacent clusters and for updating the service registry using the *UpdateInfo* message, described by Algorithms 2 and 3. We use a heartbeat broadcast message periodically sent by every node to maintain the neighbourhood information and to trigger the events *LinkAdd* and *LinkDelete*. The heartbeat is also used for the cluster setup and maintenance, replacing the *SetRoot* message for C4SD and the *Clusterhead* and *Join* messages for DMAC. The focus of our comparative simulations is the overhead induced by the *UpdateInfo* and *ServDisc* messages in dynamic environments.

During a simulation, *border effects* [45] may appear, where nodes at the area borders establish fewer links than nodes placed in the centre of the simulation area. These effects can be avoided by using the cyclic distance model for link formation. In this model, nodes at the border of the system area establish links via the borderline to the nodes located at the opposite side of the area. This setup approximates an area where nodes are distributed according to a *homogeneous Poisson point process* [64]. Similar to the DMAC performance analysis of Bettstetter [45], run static simulations for measuring the properties of the clustering structure using the cyclic distance model, where we provide results for three transmission ranges: $r = 0.1a$, $r = 0.2a$ and $r = 0.3a$.

For the analysis of SD4WSN we run dynamic experiments, where we vary both the density and the mobility of the network. For changing the network density, we take the average of the three transmission ranges considered above ($r = 0.2a$), and we vary the number of nodes. Topological changes are triggered as a result of node mobility, which determine links to be added or deleted. For changing the network mobility, we vary the percentage of mobile nodes in the network. Following a simplistic scenario of people walking and stopping, we use the random waypoint mobility model [94], where the mobile nodes move with a speed of 1m/s and, upon arrival at an intermediate point, pause for 30 seconds before restarting. Due to the initialization problems of the random waypoint mobility model, which causes a high variability of the average number of neighbours in the first part of the simulation, we follow the recommendation of Camp et al. [52] and we discard the initial 1000 seconds of simulation time in each simulation trial. We count the number of messages for the next 1000 seconds.

We study the impact of the chosen clustering algorithm on the performance of the service discovery protocol. For our analysis, we compare C4SD with DMAC. First, we study the properties of the algorithms in terms of cluster density and cluster height. Second, we measure the performance of the service discovery protocol running on both structures under the same topological conditions.

5.4.2 Properties of the clustering algorithms

In this section, we analyse two fundamental properties of clustering algorithms (i.e. the cluster density and cluster height), which have an important influence on the performance of the service discovery protocol, as shown in Section 5.4.3.

5.4.2.1 Cluster density

We define the cluster density as the expected percentage of clusterheads to the total number of nodes in the network. The cluster density is an important measure for the performance of a clustering algorithm that is intended to be used as a basis for a search mechanism. A high density of clusters leads to a large number of messages exchanged in the discovery phase.

In order to analyse the expected node degree when the border effects are not present (see Section 5.4.1 for an explanation of border effects), we consider the limiting case of a uniform distribution, denoted as a homogeneous Poisson point process with density $\rho = N/a^2$. The expected node degree D, which we term the *network density*, is [45]:

$$E\{D\} = \rho\pi r^2 = N\frac{r^2\pi}{a^2} \tag{5.1}$$

The spatial distribution of the root nodes for both clustering algorithms belongs to the family of *hard-core point processes* [149], in which the constituent points are forbidden to lie closer together than a certain minimum distance. For our clustering algorithm, we approximate the cluster density by using the *Matérn hard-core process*, where a point from a homogeneous Poisson point process is only retained if it has the highest weight of all Poisson points located within a circle around is position. The retaining probability P_{C4SD} of nodes that become roots, i.e. the cluster density, is the following:

$$P_{C4SD} = \frac{1}{E\{D\}}(1 - e^{-E\{D\}}) \tag{5.2}$$

This equation shows that when the network density approaches 0, the probability to become clusterhead tends to 1, since the node becomes isolated. If the expected degree grows to infinity, the probability of becoming a clusterhead goes to 0. Eq. 5.2 enables us to compute the expected number of clusters E_{C4SD}:

$$E_{C4SD} = P_{C4SD}N = \frac{a^2}{\pi r^2}(1 - e^{-\frac{N\pi r^2}{a^2}}) \tag{5.3}$$

For the DMAC clustering algorithm, the probability of a randomly chosen node to become clusterhead is given by Bettstetter [45]:

$$P_{DMAC} = \frac{1}{1 + \frac{E\{D\}}{2}} \tag{5.4}$$

The probability of becoming clusterhead for DMAC evolves similarly with P_{C4SD}: if the expected degree grows to infinity, the probability of becoming a clusterhead goes to 0. For an expected degree 0, the P_{DMAC} is 1.

From Eq 5.4 we can deduce the expected number of clusters in DMAC:

$$E_{DMAC} = P_{DMAC}N = \frac{1}{\frac{1}{N} + \frac{\pi r^2}{2a^2}} \tag{5.5}$$

From Eq. 5.3 and 5.5 it can be shown that:

- $P_{C4SD} < P_{DMAC}$
- for r and a fixed, the function $f(N) = E_{DMAC} - E_{C4SD}$ is strictly increasing
- $\lim_{N\to\infty} E_{DMAC} = 2\lim_{N\to\infty} E_{C4SD}$

We can conclude that: (1) C4SD has a lower cluster density, (2) the difference in the number of clusters built by the two protocols increases with the network density and (3) C4SD almost halves the total number of clusters for saturated areas.

For validating the theoretical estimation of C4SD we run simulations with three transmission ranges $0.1a$, $0.2a$ and $0.3a$, and we count the number of clusters formed in each experiment. The mean of the samples are shown in Figure 5.4, with $5th$ and $95th$ percentiles. We also plot the theoretical estimations for the three values of the transmission range. The figure shows that the estimated values match exactly the simulation results.

In the first part of the plot, the nodes are sparsely distributed on the simulation area and form clusters with only one member. When the network becomes dense, the new nodes added either join the already existing clusters or they form their own cluster and force the root nodes in the neighbourhood to join. From a certain number of nodes in the area, adding new nodes does not change the number of clusters.

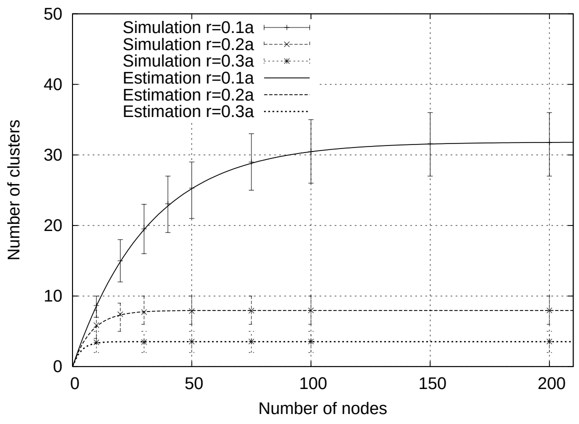


Figure 5.4: C4SD - average number of clusters on $a \times a$ area, with $5th$ and $95th$ percentiles.

5.4.2.2 Cluster height

The design of DMAC imposes a maximum cluster height of one, so in this section we are only interested to evaluate the cluster height for C4SD, which does not have a theoretical limit. We run simulations with three transmission ranges, and for each of them we vary the number of nodes. Figure 5.5 shows the results as a function of the expected node degree, with the $5th$ and $95th$ percentile values as error bars.

We can notice that for all the three transmission ranges, the points follow the same curve. We conclude that, similarly to the cluster density, the average cluster height is only a function of the expected node degree (or network density). This shows that the clustering algorithm has a good scalability performance. The second conclusion is that the average cluster height is lower than 2, and at least 95% of the clusters have the height lower or equal to 3. This result indicates that we can achieve relatively small-height clusters without imposing a maximal hop diameter limit, which would increase the maintenance effort and generate the chain reaction effects.

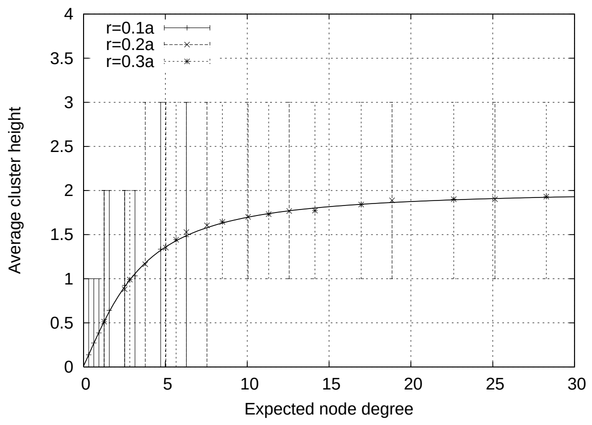


Figure 5.5: C4SD - average cluster height with $5th$ and $95th$ percentiles.

5.4.3 Service discovery performance

We compare the performance of the service discovery protocol using both DMAC and C4SD, under the same topological and mobility conditions. Due to the mentioned dissimilarities between the two protocols, we expect different behaviours when using them for discovery purposes.

We analyse the communication cost during the maintenance of the clustering structure and the discovery of services. The energy consumption for transmitting and receiving packets depends both on the hardware platform and the MAC protocol. For this reason, instead of showing the results in terms of energy consumption, we use the more general measure of counting the number of packets sent and received for maintenance and discovery. The energy consumption increases linearly with the time a node spends in transmission and reception modes, which is proportional to the number of messages sent and received [134, 63].

5.4.3.1 Maintenance overhead

In the first experiment we study the impact of the network density on the maintenance overhead (number of *UpdateInfo* messages), in an average mobile scenario, where 50% of the nodes are moving acording to the mobility model described in Section 5.4.1.

When a node moves from one cluster to another, the old service registration is deleted and a new registration is sent to the new clusterhead. However, the knowledge on adjacent clusters needs more overhead: when a node v moves from one cluster to another, all former neighbours of v must delete the information related to v and report the change to their parents, which can belong to different clusters. Similarly, all the new neighbours of v must add the information provided by v and send it to their parents. On the one hand, due to lower cluster density, C4SD has a lower overhead of maintaining the knowledge on adjacent clusters. On the other hand, the service registration is cheaper for DMAC due to the smaller cluster height. We are interested to examine the cumulative maintenance overhead with different network densities.

Figure 5.6 shows the average number of messages sent and received by a node in the network in one second of simulation time (as explained earlier, the start-up phase of dynamic simulations is ignored). For sparse networks, where there are few neighbouring clusters, the DMAC protocol behaves better. For dense networks, the effort for maintaining the knowledge of adjacent clusters dominates the overhead of service registrations, and thus C4SD overtakes DMAC. We notice that for both protocols, the number of messages remains approximately constant after a certain network density. This is due to the fact that for dense networks, it is likely that the information received from a new neighbour does not change the already acquired knowledge of adjacent clusters, so it is not propagated further in the tree.

We analyse the behaviour further when increasing the network mobility. Figure 5.7 shows the experimental results with 100 nodes and percentage of mobile nodes between 10% and 90%. We count the average number of messages per second sent and received by a node. Compared to DMAC, C4SD behaves progressively better when increasing the network mobility. The reason is that the chain reaction inherent to DMAC triggers additional maintenance overhead of the directory structure, where the service information and the knowledge on adjacent clusters have to be updated at the new clusterheads. The more dynamic the network, the more probable is this reaction to occur.

The numerical mean values of the simulation results, as well as the 5th and 95th percentiles, are given in the Appendix.

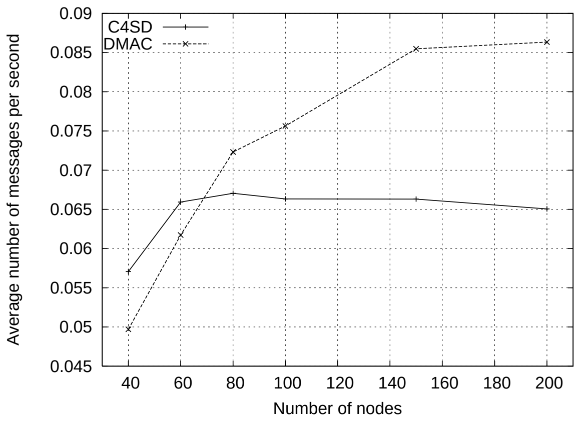


Figure 5.6: The average number of *UpdateInfo* messages sent and received per node in one second depending on the number of nodes.

5.4.3.2 Hit ratio

We analyse the hit ratio of both C4SD and DMAC, where the *ServDisc* messages are forwarded to the service provider (see Section 5.3.2). A hit is present if the service request reaches the matching service provider. The hit ratio is defined as the number of hits over the total number of service requests.

Since C4SD has on average a higher cluster height than DMAC, the convergence of service registrations is slower. As a consequence, we expect DMAC to have a better hit ratio.

In our first experiments, no caching mechanism is involved. Figure 5.8 shows the results depending on the percentage of moving nodes. As expected, DMAC performs better than C4SD due to faster convergence. However, DMAC hit

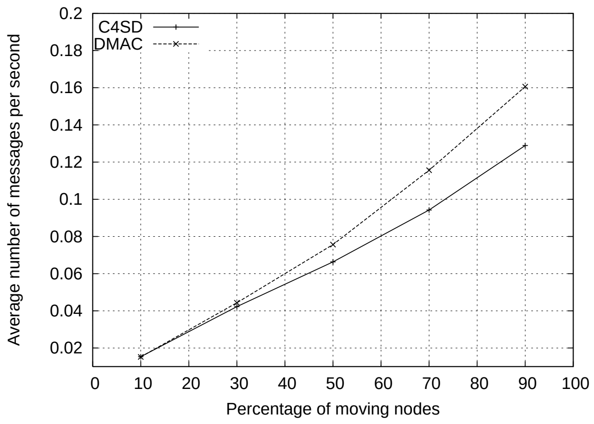


Figure 5.7: The average number of *UpdateInfo* messages sent and received per node in one second depending on the percentage of moving nodes.

ratio drops similarly when increasing the network mobility. In our second set of experiments we implement a limited-time caching of service requests (see Section 5.3.2). By implementing caching we obtain a high hit ratio for both protocols, which is above 0.98 for all mobility cases that we consider (see Figure 5.8).

5.4.3.3 The cost of service discovery

We are interested in the number of *ServDisc* messages exchanged during one service discovery phase. Since C4SD has a lower cluster degree, we expect that it also experiences a lower discovery cost. Figure 5.9 shows the average number of service discovery messages sent and received per node for a network of 100 nodes, depending on the percentage of moving nodes. We represent the cost of service discovery with and without caching. We notice that caching implies more messages spent in the service discovery phase. The discovery cost is significantly smaller for C4SD (up to 50%), due to the lower cluster density. DMAC experiences a rapid growth in the discovery cost when caching is implemented.

Figure 5.10 shows the number of service discovery messages sent and received per node, for a network with 50% of moving nodes, depending on the number

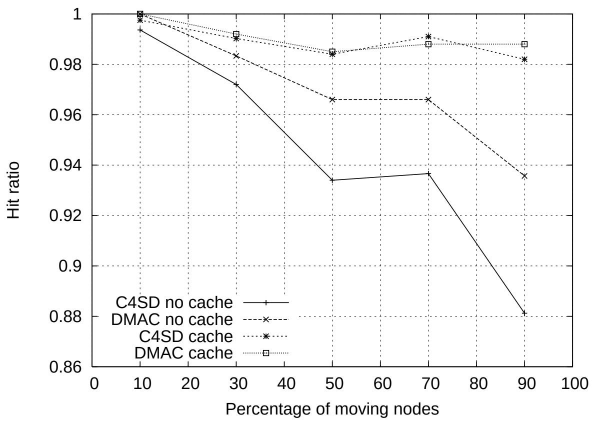


Figure 5.8: Ratio of successful service requests depending on the percentage of moving nodes.

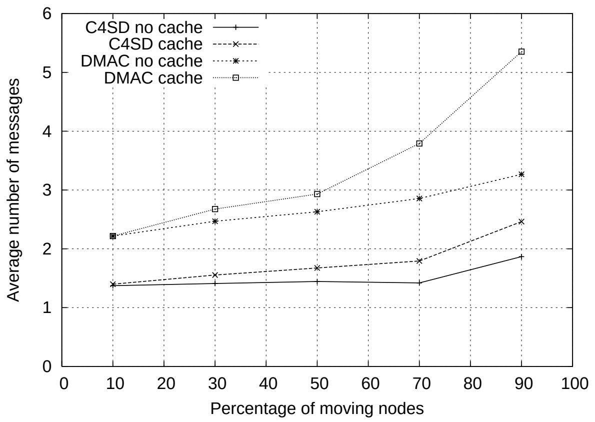


Figure 5.9: The average number of *ServDisc* messages sent and received per node depending on the percentage of moving nodes.

of nodes. For dense networks, the number of messages per node decreases when increasing the network density. This is an inherent property of our cluster-based service discovery protocol, where the discovery messages are exchanged among the root nodes. The reason is that the number of clusters converges to a finite value when the network density increases, for both C4SD and DMAC (see Section 5.4.2.1).

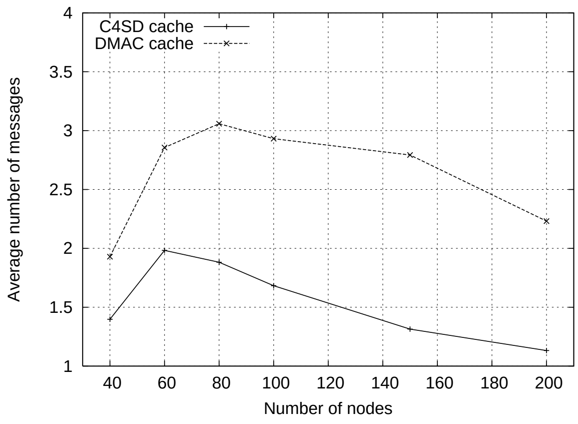


Figure 5.10: The average number of *ServDisc* messages sent and received per node depending on the number of nodes.

The numerical values of the results, as well as the 5^{th} and 95^{th} percentiles, are given in the Appendix. We observe that the 5^{th} percentile is close to 0 for both C4SD and DMAC. The reason is that when a node searches for a service present in its own cluster, the discovery process takes place locally, and thus only a few messages are exchanged until the service is found.

To conclude this section, we make the following comments:

- The lower the cluster density, the lower the maintenance, discovery and service reply costs.
- Smaller-height clusters achieve faster convergence and higher hit ratio.
- Caching of service requests can be used to improve the hit ratio.
- The chain reaction leads to higher maintenance costs.

It follows that C4SD overtakes DMAC in dense and dynamic networks, when caching is implemented for improving the hit ratio.

5.5 Implementation

In this section, we focus on practical issues when implementing the service discovery mechanism on resource-lean sensor nodes. The resulting service-oriented WSN has to fulfil the following three objectives:

1. To self-organize in a clustered structure and to detect and react to topology changes due to mobility or communication errors.
2. To offer the users the possibility to discover and use the services available in the network. The discovery protocol should exploit the underlying clustered topology for a scalable and energy-efficient search.
3. To discover and use the available gateways to the outside world (mobile phones, PDAs, laptops) that can be used for signalling relevant events, for example alarms in the case of safety-critical situations.

From a practical point of view, we show that the solution is lightweight (both code and data memory footprint), the interaction user-WSN is straightforward and intuitive and the alarming from WSN to the world is simplified by using the same discovery and communication protocol. The performance measurements validate the theoretical and simulation results and show that the clustering algorithm has low communication overhead and the service discovery protocol scales with the network density.

5.5.1 Optimizations and extended functionality

In addition to the clustering and service discovery modules, our solution integrates a series of optimizations for further reducing the energy consumption. Moreover, the functionality is extended by allowing mobile gateway nodes to join the WSN in an ad-hoc manner.

5.5.1.1 Cross-layer integration with MAC

When nodes change their role in the clustering structure, the *SetRoot* message is broadcast from each node experiencing the change to its neighbours. Via this message, the root identity is disseminated to the whole cluster and to the borderline nodes belonging to adjacent clusters. A cross-layer approach with a TDMA-based MAC protocol allows for an integration of the *SetRoot* message with the control message exchanged periodically by neighbours at the MAC layer. This method reduces the communication overhead and thus increases the network lifetime (see Section 5.5.3 for a description of the cross-layer integration with LMAC).

5.5.1.2 Integration of discovery and usage

After service discovery, the consumer is interested to make use of the desired service, by selecting and establishing a connection with the appropriate service provider. In the case where providing a service is equivalent to sending a short piece of information to the consumer, extensive message exchange can be avoided by integrating the service usage in the service discovery process. We identify the following cases:

1. *Service discovery and usage integration during the reply phase.* The service reply may incorporate partially or totally the information exchanged during service usage. For example, suppose a service consumer issues a discovery message which searches for the location of the nodes that are sensing a particular measure or detecting an event. Upon receiving this message, the nodes that match the service description issue a reply message that already includes the location coordinates. In this way, subsequent service usage dialogue is avoided.

2. *Service discovery and usage integration during the discovery phase.* In a similar manner, the discovery message may include an event, such as an alarm about an undesired situation detected in the network (for example a temperature exceeding a threshold value). In this case, the service discovery message tries to discover gateway nodes capable of announcing the event outside the network. The discovery message contains the service usage information (e.g. the sensor readings), so the service reply and usage phases may be no longer be required.

5.5.1.3 Integration with mobile platforms

The traditional model of a WSN requires one sink node that collects the sensor data and acts as a gateway to other networks such as Internet [31]. Our environment is composed of a multitude of sensor platforms and mobile devices, which form a heterogeneous network. Some sensor nodes have gateway capabilities, being connected via wireless links (Bluetooth) to Smartphones or PDAs, which in turn are connected to the GSM network or to the Internet (through WLAN). In this way, the traditional model of WSN is extended to multiple, mobile gateway nodes. Using the gateway nodes, a two-way communication is possible: (1) users can interact with the WSN by discovering and using the available services, and (2) the WSN can discover and use the gateway nodes to announce relevant events, for example alarms in the case of safety-critical situations.

We implement our service-oriented solution on sensor nodes, taking into account the optimizations mentioned in Section 5.5.1 and the multiple gateway integration. In what follows, we describe the hardware and the software details of our implementation.

5.5.2 Hardware

We implement SD4WSN on the Ambient μNode 2.0 platform [13], equipped with a low-power MSP430 micro-controller produced by Texas Instruments. The sensor node offers 48kB of Flash memory and 10kB of RAM. The radio operates in the 868/915MHz band and has a maximum data rate of 100kbps.

The connection of sensor nodes to mobile devices is done by using the Parani-ESD200 Bluetooth module [19]. This module can communicate through its on-board antenna with other Bluetooth devices in the range of 30m. It can be connected to the Ambient μNodes via standard UART interface, and configured and controlled by the typical AT command set. Figure 5.11 shows the Ambient sensor node platform and the Parani Bluetooth module.

The node provided with a Bluetooth interface can connect to any other Bluetooth-enabled device that supports the Serial Port Profile. We use Smartphones and PDAs carried by people as mobile gateways that can connect to the GSM network and the Internet.

5.5.3 Software

Figure 5.12 shows the main functional modules of our prototype implementation. For regulating the access of the sensor nodes to the wireless medium, we use LMAC [87], an energy-efficient TDMA-based MAC protocol designed for WSNs. LMAC divides time into slots, each node being assigned one slot when it can transmit the data in a collision-free wireless medium. During its time slot, a node transmits a control packet (as a heartbeat), followed by the actual data (if any). The higher-layer protocols can use the control packet to piggyback a small piece of information for an energy-efficient cross-layer integration. We use

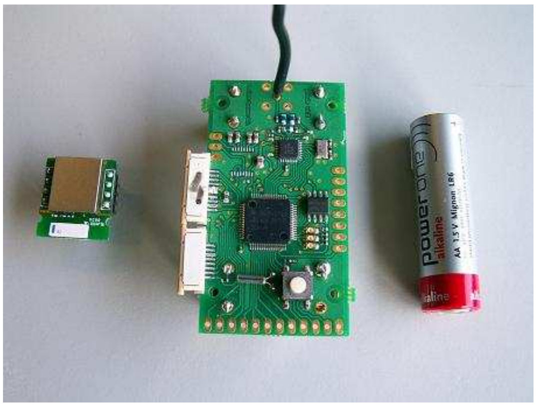


Figure 5.11: The hardware: Parani-ESD200 Bluetooth module and Ambient μNode 2.0 platform

this facility to disseminate and locally update the root identity, thus replacing the *SetRoot* message of our clustering algorithm (see Section 5.5.1).

By analysing the control messages received from the neighbours, LMAC constructs a neighbour table which is constantly maintained up-to-date. In this way, LMAC can inform the higher layer protocols of the changes in the network topology (links added or deleted). The *Clustering Topology Control* module analyses the neighbourhood information provided by LMAC and constructs the clustering structure, by choosing the appropriate parent node. Using the cross-layer optimization, this module is informed about the root identity from the control message exchanged periodically by LMAC. Through LMAC, the *Clustering Topology Control* module receives from the children nodes and transmits to the parent node the knowledge on adjacent clusters and the service registrations.

The *Service Discovery and Usage* module receives either from the neighbours, through LMAC, or from the mobile gateway, through Bluetooth, the service discovery and usage unicast messages. In order to decide the next hop for the service discovery messages, this module uses the clustering structure, the information on adjacent clusters and the service registrations provided by the *Clustering Topology Control* module. The received message is then forwarded either to a neighbour from the WSN through LMAC, or to the gateway, through Bluetooth. The *Service Discovery and Usage* module also receives the sampled values from the attached sensors, so that it can include the sensor readings in

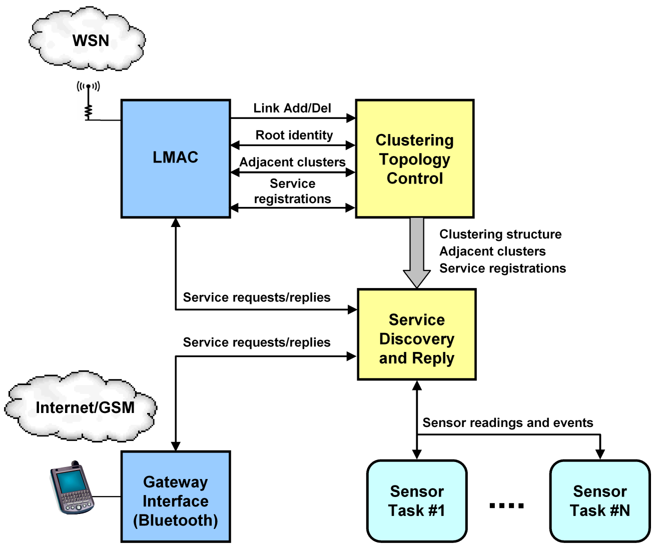


Figure 5.12: Implementation overview.

the service usage message exchange.

The SD4WSN modules, (e.g. *Clustering Topology Control Service Discovery and Usage*) have a total code memory footprint of 3KB and require 342B of RAM, considering an average network density of 32 neighbours per node.

5.5.4 Demonstration setting

We present a demonstration setting as a proof of concept for our service-oriented solution [10]. Figure 5.13 shows a set of sensor nodes organized into three clusters and connected to gateway devices via Bluetooth interfaces. The capability grades of the nodes determine the formation of clusters. The gateway nodes have the two highest capability grades (14 and 15), as they are connected to the Smartphones and PDAs. Consequently, they are chosen as clusterheads of clusters 3 and 1. The clustering structure dynamically reorganizes in case of mobility or node addition/removal.

The sensor nodes are equipped with light, temperature and movement (tilt switch) sensors (see Table 5.5.4).

The following services are available in this setting:

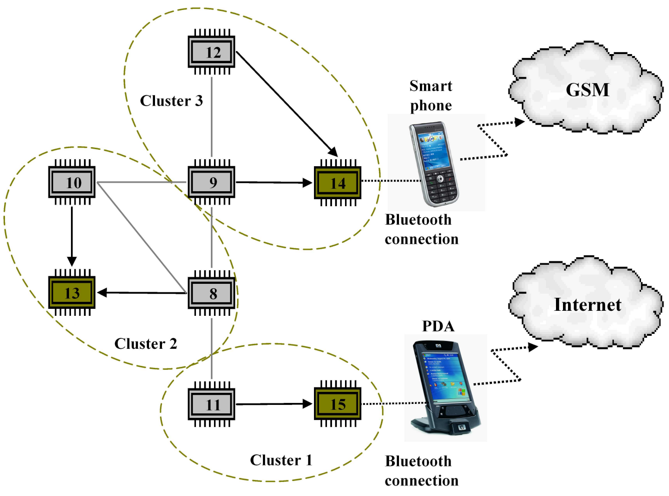


Figure 5.13: Demonstration setting.

1. *Sensor reading service* - provides a simple sensor reading at the moment of invocation (e.g. temperature reading).

2. *Monitoring service* - provides the history of a specified measure (e.g. light monitoring over the last 20 minutes). This service necessitates the establishment of a communication session between the provider and the consumer. First, the service parameters (e.g. time history) are established between the two parties. Then, the service provider delivers the desired history measurements to the user.

3. *Alert service* - enables the sending of SMS or Email through the mobile devices when an an abnormal situation occurs in the network. The WSN searches for these services provided by gateway nodes in order to announce the event to the GSM network or Internet.

Figure 5.14 shows the service discovery and usage console which runs on the mobile devices. The user can select the desired service from a list and launch the search command which initiates a service discovery message. The message is transmitted via the Bluetooth interface to the gateway node and then forwarded according to the SD4WSN protocol. When the service is found, a service reply message will announce the consumer about the result of the search.

Sensor	**Manufacturer**	**Model**
Light	Texas Advanced Optoelectronic Solutions	TSL2550
Temperature	National Semiconductor	LM92
Tilt switch	Assemtech Europe	CW1300-1

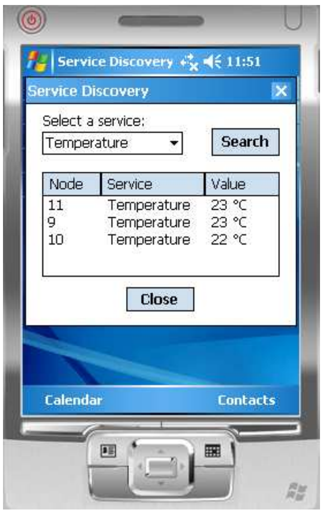
Table 5.1: Sensor types

The result of discovering a simple temperature reading service is shown in Figure 5.14(a). Besides the address of the service provider, the service reply message also incorporates the temperature value (see Section 5.5.1 for the integration of service discovery and usage). The user can browse the list of sensor nodes that offer the temperature service, having also access to the sensor readings.

In contrast to the sensor reading service, the monitoring services require a more complex dialogue between provider and consumer. First, the user launches a discovery message to find out the providers of the monitoring service. Upon receiving the list of providers, the user selects one of them, establishes the service parameters and invokes the service. Figure 5.14(b) shows an example of a light monitoring service. The user selects node 11 from the list of providers and specifies the time history (20 minutes) over which to retrieve the sampled data (the maximum time history is embedded in the service reply messages: 25 minutes for node 11). The result of the service invocation is illustrated in Figure 5.14(c), where the history of the sampled data is plotted on the mobile device.

In the case where a sensor node registers an abnormal sensor value (temperature or light level exceeded, vibrations), the node issues a service discovery message which incorporates the sensor reading. The discovery message travels the clustered WSN in search for an SMS or an Email service. Upon receiving the message, the mobile device displays an alarm and announces the event (see Figure 5.14(d)).

Figure 5.15 shows a detail of the demonstration setting, namely two sensor nodes, a mobile phone and a Smartphone. The first sensor node is equipped with a tilt switch sensor and is thus capable of detecting vibrations. The second node is provided with a Bluetooth interface and can connect to the Smartphone to send events. The situation depicted in the figure is the following: the node with a tilt switch sensor detects a high level of vibrations and issues a discovery message for an SMS service. The message travels the clustered WSN and reaches



Service Discovery

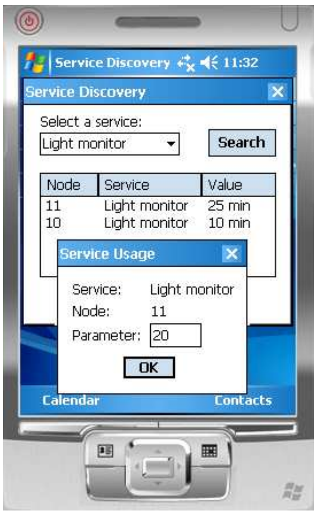
Select a service:

Temperature Search

Node	Service	Value
11	Temperature	23 °C
9	Temperature	23 °C
10	Temperature	22 °C

Close

(a) Sensor reading service usage



Service Discovery

Select a service:

Light monitor Search

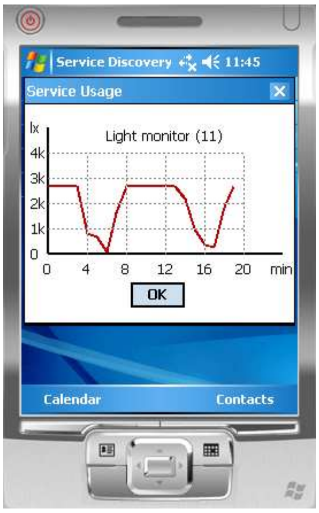
Node	Service	Value
11	Light monitor	25 min
10	Light monitor	10 min

Service Usage

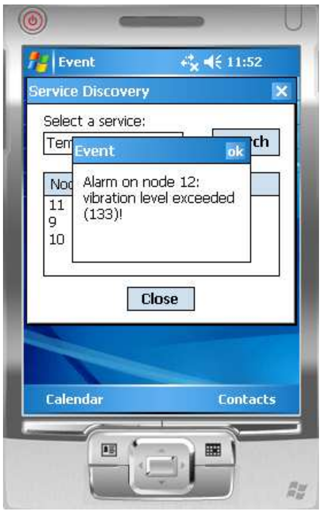
Service:	Light monitor
Node:	11
Parameter:	20

OK

(b) Monitoring service - message exchange



(c) Result of monitoring service usage



Service Discovery

Select a service:

Tem ch

Event ok

Alarm on node 12: vibration level exceeded (133)!

Noc
11
9
10

Close

(d) Gateway discovery for announcing events

Figure 5.14: Service discovery and usage console.

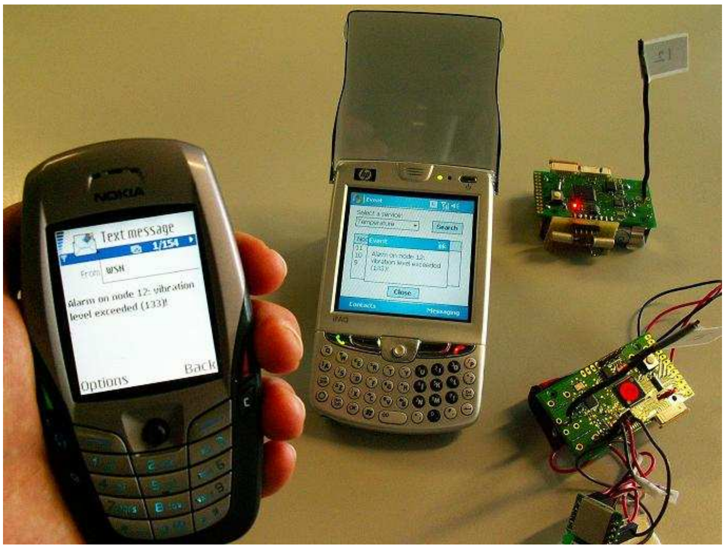


Figure 5.15: Announcing events.

the node with the Bluetooth module, which provides the desired service and transmits the message to the Smartphone. The Smartphone in turn, displays an alarm and sends an SMS to the mobile phone.

5.5.5 Performance measurements

We implement SD4WSN on the Ambient μNodes and we measure the performance by deploying a series of testbeds of up to 25 nodes. We analyse the number of messages exchanged until network convergence, the number of resulting clusters and the cost of service discovery, on an average case scenario. In order to estimate the average case, we generate a series of random topologies offline (up to 5 hops), which we disseminate to the WSN at the beginning of each test. Similar to the simulations, the nodes are considered to be placed on a square area of size $a \times a$, using the cyclic distance model. We make the observation that the cyclic distance model is not valid in a real-life setting. However, for measuring the performance of SD4WSN in a small-scale test bed, this model has the advantage that it increases the connectivity of the network.

To be able to measure the performance of the proposed algorithms under different network densities, we choose $r = 0.3a$ as the highest transmission range considered in Section 5.4. We vary the number of nodes from 5 to 25 and thus the network density (e.g. average node degree) increases from 1.4 to 7. For each network density in turn we run 20 experiments, each consisting of the following steps:

1. Disseminate the network topology.
2. Compute the number of messages exchanged until network convergence.
3. Count the number of resulting clusters.
4. Issue a service discovery message from a random node in the network.
5. Calculate the number of service discovery messages exchanged.

A gateway node connected through the serial port to a computer is used for disseminating the network topology. Each node thus receives a set of valid neighbours, while the rest of the neighbours become ignored by the MAC layer. The gateway node is also used for gathering the experimental results, by acting as a network sniffer. In this way, we can establish when nodes do not exchange messages any longer, which means that the network reached convergence or the experiment is finished.

In the following, we compare the experimental results with the simulations. To provide a fair comparison, the simulation results in this section are obtained under similar conditions as the experiments, following the steps (1)-(5).

In our experiments, the average number of resulted clusters lies between 3 and 3.5, which is in accordance with the theoretical and simulation results from Figure 5.4.

Figure 5.16 shows the average number of *UpdateInfo* messages sent and received per node for updating the knowledge on adjacent clusters and the service registrations. The messages are counted starting from a newly initialized network until the network convergence. We notice that the results of the implementation are close to the simulation results. In the first part of the plot, the number of messages per node grows linearly with the network density, while in the last part of the plot, the increase in the number of messages per node starts attenuating. The same result is obtained for our dynamic simulations (see Figure 5.6).

We represent in Figure 5.17 the number of service discovery messages sent and received per node during one service discovery phase. We notice the similarity between the implementation and the simulation results: once the network

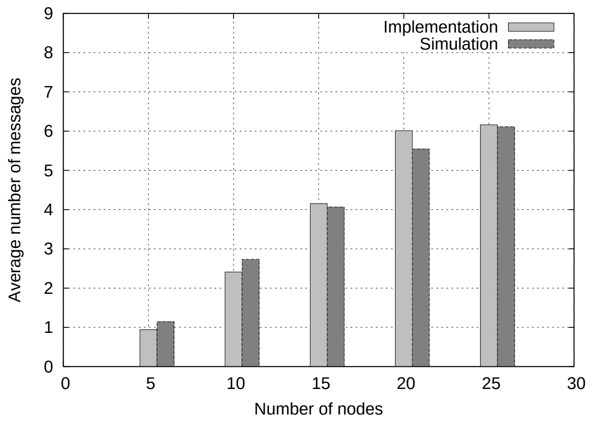


Figure 5.16: The average number of *UpdateInfo* messages sent and received per node until network convergence.

is connected, increasing the density does not increase total discovery cost, because the service discovery messages travel only among the clusterhead nodes, which remain constant in number. Therefore, in the second part of the plot, the total number of discovery messages remains on average approximately constant, which means that the average number of discovery messages sent and received per node decreases. A similar evolution is found for the dynamic simulations (see Figure 5.10).

The numerical values of the implementation results with 5^{th} and 95^{th} percentiles are given in the Appendix. Similar to the results from Section 5.4.3.3, the 5^{th} percentile of the discovery cost is close to 0, due to the low overhead induced by a local search (i.e. within the same cluster).

5.6 Conclusions

This chapter proposes SD4WSN, a service discovery protocol for heterogeneous WSNs. The protocol relies on C4SD, a clustering algorithm that offers distributed storage of service descriptions. The clusterhead nodes act as directories for the services in their clusters. The structure ensures low construction and maintenance overhead, reacts quickly to topology changes and avoids the chain-reaction problems. The topology is not required to be static during the

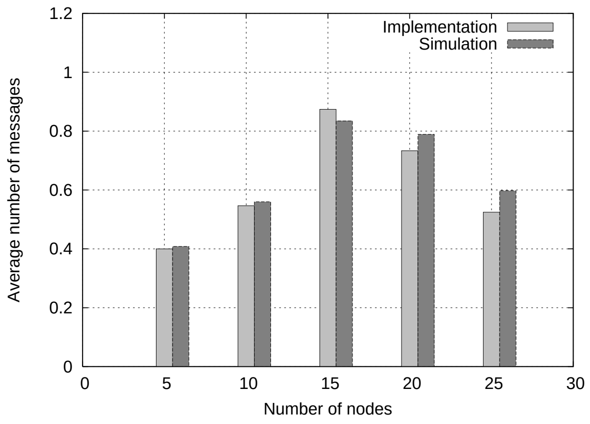


Figure 5.17: Average number of *ServDisc* messages sent and received per node during one service discovery phase.

cluster initiation phase. A service lookup results in visiting only the clusterhead nodes, which ensures a low discovery cost.

We analyse theoretically and through simulations the properties of the clustering structure in comparison with DMAC and we investigate how the service discovery performance depends on the underlying clustering structure. Our results show that the chain reaction of DMAC determines reclustering and reregistration of services with new clusterheads, implying higher maintenance overhead. The smaller-height clusters of DMAC leads to higher hit ratio. The hit ratio is improved to more than 98% for both protocols if a mechanism of limited-time caching of service discovery messages is used. Due to the lower cluster density, C4SD has a lower discovery cost in both implementation alternatives.

We implement the proposed algorithms on resource-constraint sensor nodes and we show that the solution is lightweight (both code and data memory footprint). We evaluate the performance by running the protocol on a series of testbeds. The experimental results confirm the simulations and show that the protocol scales with the network density.

Appendix

We present the numerical average values from simulations and experiments, along with 5th and 95th percentiles, in order to complement the point estimations with confidence intervals.

Number of nodes	**40**	**60**	**80**	**100**	**150**	**200**
C4SD - average value	0.057	0.066	0.067	0.066	0.066	0.065
C4SD - 5th percentile	0.049	0.058	0.058	0.057	0.059	0.058
C4SD - 95th percentile	0.066	0.077	0.077	0.072	0.073	0.072
DMAC - average value	0.05	0.062	0.072	0.076	0.085	0.086
DMAC - 5th percentile	0.043	0.054	0.062	0.06	0.07	0.068
DMAC - 95th percentile	0.057	0.069	0.084	0.09	0.105	0.108

Table 5.2: Average number of *UpdateInfo* messages sent and received per node in one second, depending on the number of nodes, with 5th and 95th percentiles (see Figure 5.6)

Percentage of moving nodes	**10**	**30**	**50**	**70**	**90**
C4SD - average value	0.015	0.042	0.068	0.096	0.13
C4SD - 5th percentile	0.012	0.035	0.059	0.086	0.122
C4SD - 95th percentile	0.017	0.047	0.078	0.109	0.138
DMAC - average value	0.014	0.043	0.076	0.118	0.161
DMAC - 5th percentile	0.012	0.036	0.06	0.099	0.15
DMAC - 95th percentile	0.016	0.054	0.09	0.137	0.172

Table 5.3: Average number of *UpdateInfo* messages sent and received per node in one second, depending on the percentage of moving nodes, with 5th and 95th percentiles (see Figure 5.7)

Percentage of moving nodes	**10**	**30**	**50**	**70**	**90**
C4SD no cache - average value	1.37	1.41	1.44	1.42	1.87
C4SD no cache - 5^{th} percentile	0.04	0.04	0.04	0.04	0.04
C4SD no cache - 95^{th} percentile	2.14	2.22	2.36	2.5	3.94
C4SD cache - average value	1.42	1.55	1.69	1.86	2.48
C4SD cache - 5^{th} percentile	0.04	0.04	0.04	0.04	0.04
C4SD cache - 95^{th} percentile	2.2	2.46	2.7	3.18	4.7
DMAC no cache - average value	2.22	2.47	2.63	2.86	3.27
DMAC no cache - 5^{th} percentile	0.04	0.04	0.04	0.04	0.04
DMAC no cache - 95^{th} percentile	3.28	3.88	4.52	5.08	5.66
DMAC cache - average value	2.29	2.7	3.11	3.84	5.33
DMAC cache - 5^{th} percentile	0.04	0.04	0.04	0.04	0.04
DMAC cache - 95^{th} percentile	3.31	4.1	5.18	6.69	9.46

Table 5.4: Average number of *ServDisc* messages sent and received per node, depending on the percentage of moving nodes, with 5^{th} and 95^{th} percentiles (see Figure 5.9)

Number of nodes	**40**	**60**	**80**	**100**	**150**	**200**
C4SD cache - average value	1.4	1.98	1.88	1.68	1.31	1.13
C4SD cache - 5^{th} percentile	0	0.02	0.04	0.04	0.02	0.02
C4SD cache - 95^{th} percentile	2.71	3.2	2.98	2.86	2.73	2.16
DMAC cache - average value	1.93	2.86	3.06	3.11	2.79	2.23
DMAC cache - 5^{th} percentile	0	0.03	0.05	0.04	0.03	0.02
DMAC cache - 95^{th} percentile	3.95	4.85	4.93	5.18	4.7	3.45

Table 5.5: Average number of *ServDisc* messages sent and received per node, depending on the number of nodes, with 5^{th} and 95^{th} percentiles (see Figure 5.10)

Number of nodes	**5**	**10**	**15**	**20**	**25**
Implementation - average value	0.94	2.41	4.15	6.09	6.35
Implementation - 5th percentile	0.4	1.2	2.27	4.29	4.93
Implementation - 95th percentile	2	3.92	6.58	7.54	8.18
Simulation - average value	1.14	2.73	4.07	5.52	6.11
Simulation - 5th percentile	0.4	1.6	2.67	3.6	4.11
Simulation - 95th percentile	2.8	4.2	6.64	7.9	8.29

Table 5.6: Average number of *UpdateInfo* messages sent and received per node until network convergence, with 5th and 95th percentiles (see Figure 5.16)

Number of nodes	**5**	**10**	**15**	**20**	**25**
Implementation - average value	0.4	0.55	0.87	0.73	0.52
Implementation - 5th percentile	0	0	0.06	0.05	0.07
Implementation - 95th percentile	0.5	0.8	1.4	1.01	0.77
Simulation - average value	0.41	0.56	0.83	0.79	0.6
Simulation - 5th percentile	0	0	0	0.05	0.01
Simulation - 95th percentile	0.53	0.86	1.11	0.93	0.8

Table 5.7: Average number of *ServDisc* messages sent and received per node during one service discovery phase, with 5th and 95th percentiles (see Figure 5.17)

Chapter 6

On-line recognition of joint movement in wireless sensor networks

We propose a method through which dynamic sensor nodes determine that they move together or separately by communicating and correlating their movement information. This method provides the necessary information to construct clusters based on semantic similarity among sensor nodes. We describe two possible solutions, one using inexpensive tilt switches, and another one using low-cost MEMS accelerometers. We implement a fast, incremental correlation algorithm, which can run on resource constrained devices. The tests with the implementation on real sensor nodes show that the method distinguishes between joint and separate movements. In addition, we analyse the scalability from four different perspectives: communication, energy, memory and execution speed. The solution using tilt switches proves to be simpler, cheaper and more energy efficient, while the accelerometer-based solution is more accurate and more robust to sensor alignment problems.

6.1 Introduction

Emerging applications of wireless sensor networks (WSNs) demand an increasing degree of *dynamics*. The sensor nodes are expected to take decisions autonomously, by using *context-aware reasoning*, and to provide an overall solution that is more reliable, accurate and responsive than traditional approaches. Examples of recent application domains include industrial processes, transport and logistics, user guidance in emergency situations [121]. In all these scenarios, we notice a growing interest in having many small, cheap devices that self-organize and cooperate, in order to supervise and actively support the actual processes. The challenges shift, accordingly, from small-scale user-to-device interaction towards large-scale device-with-device *collaboration*. In parallel, the design choices migrate from centralized approaches towards scalable, distributed techniques, that can be implemented on resource constrained devices.

Our goal is to develop a method for constructing dynamic clusters based on nodes sharing a common context. We argue that such a method opens perspectives for a large variety of applications, ranging from user entertainment (people hiking or skiing together) and healthcare (body area networks), to smart vehicles carrying smart goods (in the field of transport and logistics, as we describe in Section 6.2). In this chapter, the common context is the movement information. More specifically, we consider a set of nodes being *together* if their movement *correlates* for a certain amount of time. Nevertheless, correlating the movement information raises a number of questions:

1. How to extract and communicate the movement information?
2. How to compute the correlation, taking into account the resource limitations of the sensor nodes?
3. How does the method scale with the number of nodes?
4. How accurate is the solution and which are the benefits and limitations?

The contribution of this chapter is a lightweight, fast and cheap method for correlating the movement data among sensor nodes, for the purpose of clustering nodes moving together. Each node correlates the movement data generated by the local movement sensor with the movement data broadcast periodically by its neighbours. The result of the correlation is a measure of the confidence that one node shares the same context with its neighbours, for example that they are placed in the same car. We focus in this chapter on correlating sensor nodes carried by vehicles on wheels.

We describe two possible practical solutions, one using tilt switches, and another one using MEMS accelerometers. In order to answer the aforementioned questions in detail, we analyse the scalability from several different perspectives

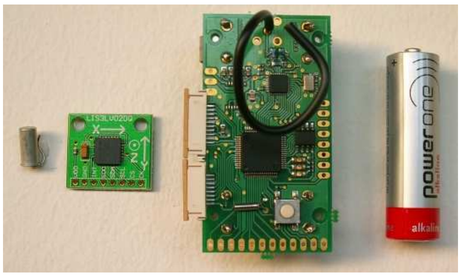


Figure 6.1: Movement sensors (left) and sensor node platform (right).

(communication, energy, memory and execution speed), and discuss the most relevant advantages and limitations. The analysis is based on the experimental results obtained from testing with real sensor nodes. We use the Ambient μNode 2.0 platform [13] with the low-power MSP430 micro-controller produced by Texas Instruments, which offers 48kB of Flash memory and 10kB of RAM. The radio transceiver has a maximum data rate of 100kbps. Figure 6.1 shows the sensors used for extracting the movement information and the sensor node platform.

In the following section we describe a concrete application setting in the field of transport and logistics, which best illustrates the idea of movement-based group awareness. Section 6.3 overviews the relevant related work. The general correlation method is described in Section 6.4. In Sections 6.5 and 6.6 we present the two practical solutions for determining the movement characteristics. Section 6.7 covers the analysis, advantages and limitations of both solutions, giving also comparative details whenever relevant. Finally, Section 6.9 formulates the conclusions.

6.2 Application setting

The motivating application is in the field of transport and logistics, which is described in detail in Chapter 1. Referring back to this application, the solution that we propose targets two specific problems regarding loading errors (products loaded into RTIs and RTIs loaded into trailers). First, the goods from an RTI correlate their movement as the RTI is pushed, and report as a group to the device carried by the order picker. In this way, a missing or wrong item can be detected before arriving on the expedition floor. Second, any RTI placed in the wrong trailer should be signalled as the truck approaches the exit gate (see Figure 1.1). Since the distance between two RTIs or two trucks is quite short, the localization of the goods inside the RTI, or of the RTIs inside the truck cannot be done reliably with radio signal strength proximity techniques. However, we consider it highly probable that two different vehicles move differently in a certain time interval. Therefore, our goal is to group the nodes based on the similarities and differences in the data generated by the movement sensors.

6.3 Related work

In the project Smart-Its, Gellersen et al. [89] formulate the notion of context sharing. The idea is to associate two smart objects by shaking them together. As a result, the user can establish an application-level connection between two devices by imposing a brief, similar movement. For example, the two devices can authenticate using secret keys that are generated based on the movement data [123]. In our work, we are interested in extending the idea of "moving together" at the group level, within large-scale transport and logistics scenarios. Therefore, we propose a fast algorithm that correlates the movement over a larger time history, and analyse the accuracy, scalability, performance and limitation factors.

Lester et al. [109] use accelerometer data to determine if two devices are carried by the same person. Human locomotion represents a repeated activity that makes an analysis in the frequency domain possible. The authors use a coherence function to derive whether the two signals are correlated at a particular frequency. Our application domain poses, however, quite different challenges. There is no regularity in the movement of the RTIs that can facilitate an analysis in the frequency domain. Moreover, the computations involved in the frequency analysis can easily overcome the resources available on sensor nodes.

The Sensemble system [36] is meant to capture the expressive motion of a dance ensemble. Sensor nodes equipped with 6-axis inertial measurement units are worn at the wrists and ankles of dancers. The movement data is transferred at high speeds (1Mbps) towards a central computer, where a cross-covariance analysis is performed, in order to express the similarity of gestures and generate a musical feedback. Our solution is different, in the sense that sensor nodes compute the correlation online, autonomously. In addition, due to price limitations, we utilize low data rate radios and just one movement sensor per node.

Lam et al. [105] propose an algorithm for dynamic grouping based on the position and speed of mobile devices equipped with GPS sensors. Nodes within a certain area that move together (similar speed and direction) form a group. However, equipping each node with a GPS sensor is not a viable solution for WSNs, because of price and power consumption considerations.

6.4 General Method

We propose a method that distinguishes between joint and separate movements of sensor nodes based on the correlation coefficient computed on two sets of data generated by movement sensors. The correlation coefficient measures the similarity between two signals by giving a number on a scale from -1 and 1. In the following, we describe in detail the practical method for computing the correlation coefficient on sensor nodes.

6.4.1 Computing the Correlation

Let **x** be one of the sensor nodes, which receives the sampled movement data from another sensor node **y**. Node **x** stores the latest sample values produced by the local movement sensor in a circular buffer X_C of size k. The buffer X_C is periodically transmitted to the neighbours at intervals $k\Delta t$, where Δt is the sampling interval. At step $i \geq 1$, **x** receives from **y** the buffer $Y_i = \{y_{(i-1)k+1}, y_{(i-1)k+2}, ..., y_{ik}\}$. Node **x** then copies the buffer X_C into a working copy $X_i = \{x_{(i-1)k+1}, x_{(i-1)k+2}, ..., x_{ik}\}$ and calculates the correlation coefficient over the last n sequences of data X_i and Y_i. More precisely, at each step i, the correlation coefficient is calculated over a sliding window of size $N = nk$, with the data $X = (x_{(i-n)k+1}, x_{(i-n)k+2}, ..., x_{ik})$ and $Y = (y_{(i-n)k+1}, y_{(i-n)k+2}, ..., y_{ik})$. Note that for $j \leq 0$, $x_j = y_j = 0$. If we denote the means of X and Y as $\bar{X}$ and $\bar{Y}$, respectively, the correlation coefficient is computed as follows:

$$\rho(X,Y) = \frac{cov(X,Y)}{\sqrt{var(X)\ var(Y)}} = \frac{\sum_{j=(i-n)k+1}^{ik}(x_j-\bar{X})(y_j-\bar{Y})}{\sqrt{\sum_{j=(i-n)k+1}^{ik}(x_j-\bar{X})^2\sum_{j=(i-n)k+1}^{ik}(y_j-\bar{Y})^2}} \tag{6.1}$$

Table 6.1 shows the execution time for computing the correlation coefficient on one sensor node, with two sets of samples of size $N = 128$. We conclude that using the direct computation from Eq. 6.1 generates slow execution times, so it is not feasible for implementation on resource-constraint devices. Therefore, we propose a fast algorithm that updates the correlation coefficient at each step. For large data sequences (large k), the memory consumption is also reduced

Method	**Complexity**	**Impl.**	**Operation**	**Time** [ms]
Direct computation	$O(N)$	float	Average	68.91
			Variance, covariance, correlation coefficient	275.65
			Total	**344.56**
Incremental algorithm	$O(k)$	int16, int32	Auxiliary sums	0.81
			Average, variance, covariance, correlation coefficient	5.47
			Total	**6.28**

Table 6.1: Execution times on MSP430 microcontroller, for N=128, k=16.

by storing only intermediate values (see Section 6.7.2 for an evaluation of the memory consumption).

The algorithm is the following. At step i, node $\mathbf{x}$ receives the buffer Y_i from node $\mathbf{y}$. Node $\mathbf{x}$ then calculates the following sums:

$$S_i^x = \sum_{j=(i-1)k+1}^{ik} x_j \quad \text{and} \quad S_i^y = \sum_{j=(i-1)k+1}^{ik} y_j \tag{6.2}$$

$$\sigma_i^x = \sum_{j=(i-1)k+1}^{ik} x_j^2 \quad \text{and} \quad \sigma_i^y = \sum_{j=(i-1)k+1}^{ik} y_j^2 \tag{6.3}$$

$$S_i^{xy} = \sum_{j=(i-1)k+1}^{ik} x_j y_j \tag{6.4}$$

Afterward, node $\mathbf{x}$ computes the following values:

$$\bar{X}_i = \bar{X}_{i-1} + \frac{S_i^x - S_{i-n}^x}{N} \tag{6.5}$$

$$\bar{Y}_i = \bar{Y}_{i-1} + \frac{S_i^y - S_{i-n}^y}{N}$$

$$var_i(X) = var_{i-1}(X) + \frac{\sigma_i^x - \sigma_{i-n}^x}{N} - (\bar{X}_i^2 - \bar{X}_{i-1}^2) \tag{6.6}$$

$$var_i(Y) = var_{i-1}(Y) + \frac{\sigma_i^y - \sigma_{i-n}^y}{N} - (\bar{Y}_i^2 - \bar{Y}_{i-1}^2)$$

$$cov_i(X,Y) = cov_{i-1}(X,Y) + \frac{S_i^{xy} - S_{i-n}^{xy}}{N} - (\bar{X}_i\bar{Y}_i - \bar{X}_{i-1}\bar{Y}_{i-1}) \tag{6.7}$$

Finally, node **x** computes the new value of the correlation coefficient:

$$\rho_i(X,Y) = \frac{cov_i(X,Y)}{\sqrt{var_i(X)\ var_i(Y)}} \tag{6.8}$$

We make the following observations:

- For all $j \leq 0$, S_j^x, S_j^y, σ_j^x, σ_j^y, S_j^{xy}, $var_j(X)$, $var_j(Y)$ and $cov_j(X,Y)$ are 0.
- If $var_i(X) = 0$ and $var_i(Y) = 0$, we take $\rho_i(X,Y) = \rho_{i-1}(X,Y)$.
  If $var_i(X) = 0$ and $var_i(Y) \neq 0$ or the other way around, we decrease $\rho_i(X,Y)$ with a value proportional to the positive variance.

Our implementation of the incremental algorithm is done on integers, because (1) it has a faster execution, and (2) the accumulation of rounding errors, which can affect the algorithm accuracy, is avoided. For integer computation, we have the following formulae:

$$N^2 var_i(X) \;=\; N^2 var_{i-1}(X) + N(\sigma_i^x - \sigma_{i-n}^x) - ((N\bar{X}_i)^2 - (N\bar{X}_{i-1})^2)$$

$$N^2 cov_i(X,Y) \;=\; N^2 cov_{i-1}(X,Y) + N(S_i^{xy} - S_{i-n}^{xy}) - N^2(\bar{X}_i\bar{Y}_i - \bar{X}_{i-1}\bar{Y}_{i-1})$$

Further details on the binary representation of data are given in the Appendix. Only the final computation of the correlation coefficient is done on floats, which does not affect the precision of the coefficient at the next step.

The algorithm proves to be much faster than the direct computation of Eq. 6.1, as shown in Table 6.1. This result makes the implementation of the online correlation on sensor nodes possible.

6.4.2 Experimental Setting

We perform two types of experiments, in which we test the proposed method:

1. The first type of experiment is intended to reproduce the movement pattern of the smart goods, in which items equipped with sensor nodes are placed in RTIs maneuvered by people. Throughout the tests, we use two RTIs on wheels, which we push on a flat surface in a sequence of start-stop movements (see Figure 6.2(a)). The characteristics of the movement are the following:

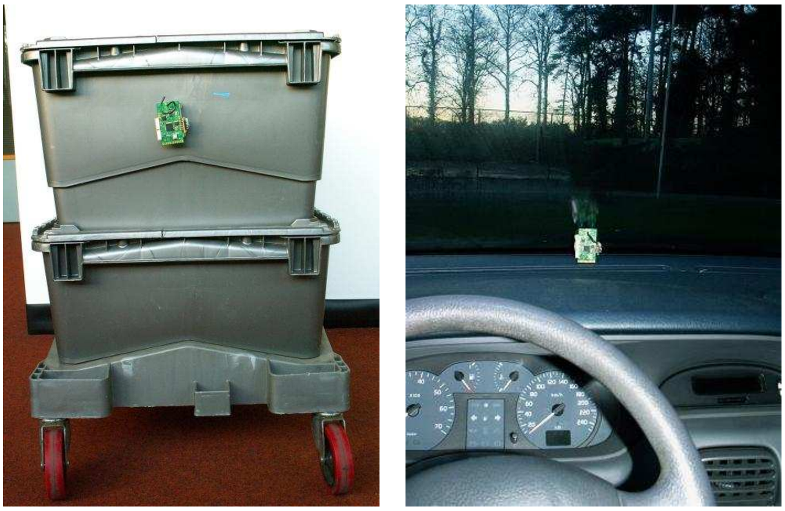

(a) Node with tilt switch attached to an RTI. (b) Node with tilt switch placed inside the car.

Figure 6.2: Sensor nodes attached to vehicles on wheels.

- Walking speed
- Mostly straight lines
- Start-stop intervals of variable size
- Sometimes reverse movements

For detecting joint movement, two sensor nodes are placed on the same RTI, while for separate movement, each sensor node is placed on a different RTI, each pushed by a different person.

2. The second type of experiment maps to the setting where RTIs are loaded into and carried by trucks. We use instead two regular cars which we drive in the university campus (see Figure 6.2(b)). Our experiments include the following types of movements:

   - Normal driving
   - Accelerating and breaking
   - Forward and backward maneuvers, curves
   - Driving on even and uneven surfaces

Parameter	**Explanation**	**Value**
k	Size of current data sequence	16 (2s)
n	No. of data sequences in data queue	8
$N = nk$	Size of queue	128
Δt	Time unit (sampling interval)	125ms
$T = N\Delta t$	Time history	16s

Table 6.2: Experimental parameters for correlating the movement data.

Two sensor nodes are placed in the same car for joint movement, while for separate movement nodes are placed inside different cars.

Each experiment lasts approximately 10 minutes. The sensor nodes broadcast the movement data together with the correlation coefficient calculated locally. A gateway node logs the coefficients and the samples from both sensor nodes to a computer through a serial interface.

6.4.3 Parameters

Table 6.2 lists the values of the parameters used in the experiments, which are chosen considering the platform constraints (sampling interval Δt and data size k) and the scenario particularities (time history T).

6.4.4 Synchronization

The synchronization between two sets of data to be correlated is important for accurately calculating the correlation coefficient. The usual synchronization mechanism requires periodic beaconing among nodes. The synchronization error is typically in the order of microseconds, depending on the sent time, access time, propagation time and receive time [68, 87]. Without implementing an explicit synchronization mechanism, our method achieves *implicit synchronization* between pairs of nodes through periodic broadcast of movement data. We use cross-layer optimization for minimizing the sent time, access time and receive time: at the moment the sender is allowed to transmit in its slot, it copies the last k samples and broadcasts them to the neighbours. When receiving the data, the last k samples of the local node are copied in a working buffer for processing. If a transmission error occurs, the next message contains again the latest k samples, such that the neighbours receive the most up-to-date sampled data.

6.5 Solution I - Tilt Switches

A ball-contact tilt switch (also referred to as ball switch or tilt switch) is a simple and cheap sensor, used in a large range of applications for coarse movement detection. Usually, the sensor is expected to provide binary information on the status of the device it is attached to (e.g. stationary/moving).

6.5.1 Extracting the Movement Information

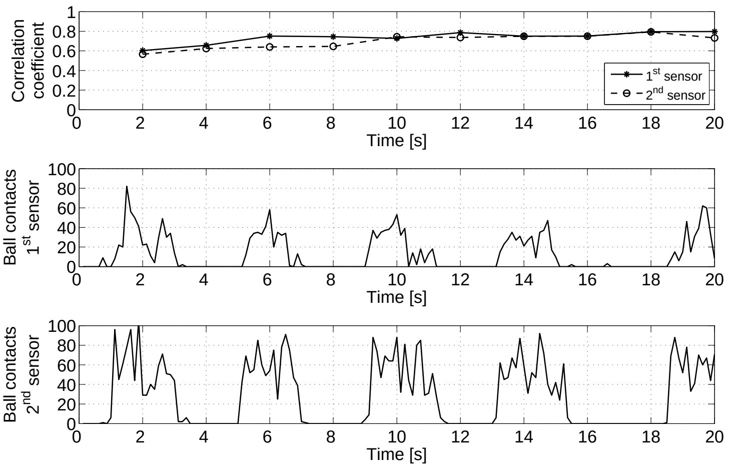
In our experiments, we are using the ASSEMTECH CW1300-1 tilt switch [14]. The price is below 2 EUR and the power consumption is approximately 2μW. Our solution is based on counting the number of contacts made by the switch ball per time unit (i.e. 125ms), as the object is moved. We make the following observations:

1. It is possible to distinguish the starting and stopping states (acceleration and deceleration) from the constant movement.
2. The tilt switch has a sensitive axis. Therefore, the sensitivity depends on the orientation of the tilt switch (see Figure 6.2 for the actual placement).
3. The results are reproducible with other switches of the same type. Although the actual values vary due to the inherent sensitivity differences and imperfect alignment of the sensitive axis, the movement pattern remains similar.

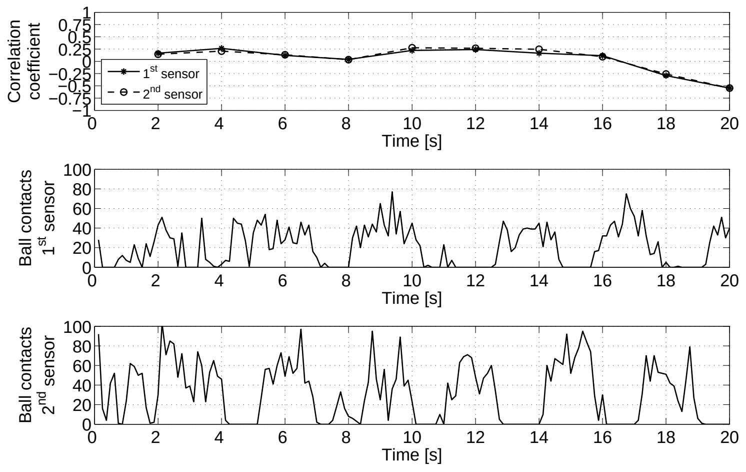
6.5.2 Experimental Results

Figure 6.3 shows the typical behaviour of the algorithm for joint and separate movements, over a period of 20 seconds. The plots at the top of the figures show the correlation coefficients calculated by the sensor nodes over the time history T, while the two bottom plots show the sampled data from the tilt switches. We make the following observations:

- The sampled data from Figure 6.3(a) show a pattern, corresponding to the alternate stationary and movement periods.
- There is a clear distinction between the moving and stationary cases: when the sensor nodes are static, the number of ball contacts is 0. Therefore, in a static situation, nodes may not need to send the whole movement buffer, but just a short indication of their state, saving thus energy.

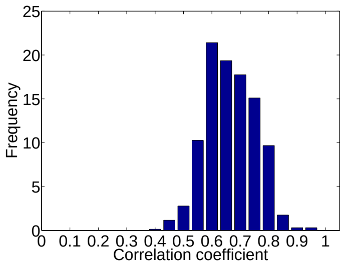


(a) Two nodes with tilt switches moving together.

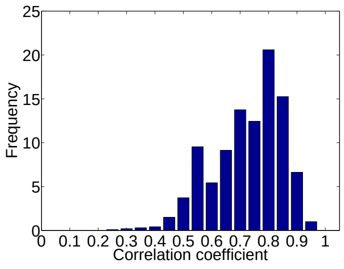


(b) Two nodes with tilt switches moving separately.

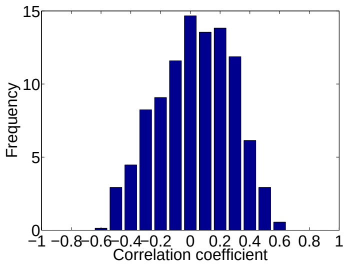
Figure 6.3: A typical behaviour of the algorithm for joint and separate movements, using tilt switches.



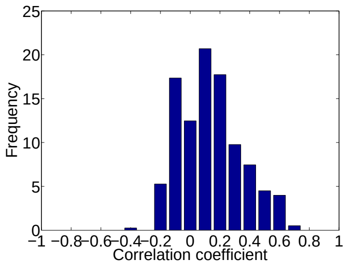
(a) Tilt switches on RTIs, moving together.



(b) Tilt switches in cars, moving together.



(c) Tilt switches on RTIs, moving separately.



(d) Tilt switches in cars, moving separately.

Figure 6.4: Histograms of correlation coefficients for tilt switches.

- For the type of movement described in Section 6.4.2, the method is successful in distinguishing between correlated and uncorrelated movements. Our experiments indicate that a high correlation coefficient corresponds to joint movement, (Figure 6.3(a)), while a low correlation coefficient is recorded during separate movements (Figure 6.3(b)).

We represent the histograms of the correlation coefficients obtained on the entire duration of the experiments (≈10 minutes) in Figure 6.4, normalized to a percentage scale. We notice the difference between the correlation coefficients computed when sensors move together (Figures 6.4(a) and 6.4(b)) and separately (Figures 6.4(c) and 6.4(d)). A more detailed analysis of the results is given in Section 6.7.1.

6.6 Solution II - Accelerometers

MEMS accelerometers have become increasingly popular recently, due to their relatively low price compared with the performance offered. The range of applications is quite broad, from movement or free-fall detection to gaming or virtual reality, and inertial navigation systems (INS) [153]. The operating principle is based on measuring the displacement of a proof mass when an acceleration is applied. The accelerometer measures, therefore, the applied acceleration (including gravitation), and outputs the values of the projections along its sensitive axis.

6.6.1 Extracting the Movement Information

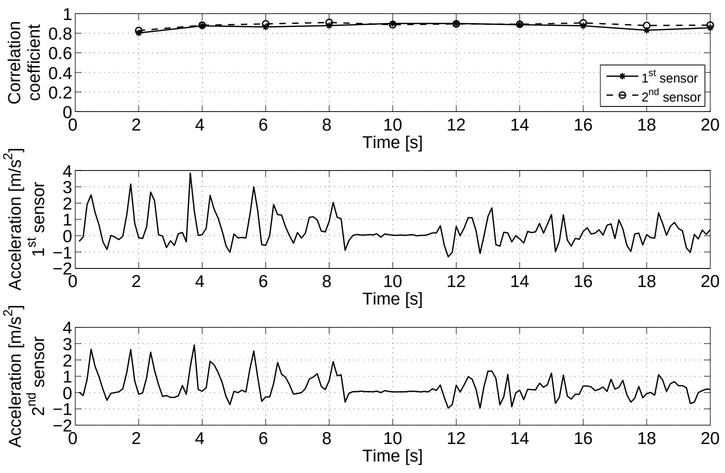
By using accelerometers, it is possible to extract elaborate information about the movement, such as the speed and distance. However, to calculate the speed and position accurately, information provided by gyroscopes has to be used for maintaining an absolute positional reference. In this way, the overall complexity and price of the system increase significantly. Moreover, the accumulation of errors require elaborated filtering and prediction techniques [135].

From these considerations, it appears that the resource-constraint sensor nodes are not yet capable of extracting and correlating speed or distance information. Therefore, we propose a simplified solution, which considers the magnitude of the acceleration vector $\parallel \mathbf{a} \parallel = \sqrt{a_x^2 + a_y^2 + a_z^2}$. The reason is that the magnitude of the sensed acceleration is the same in any frame of reference. Consequently, the alignment and orientation of the sensors are no longer important.

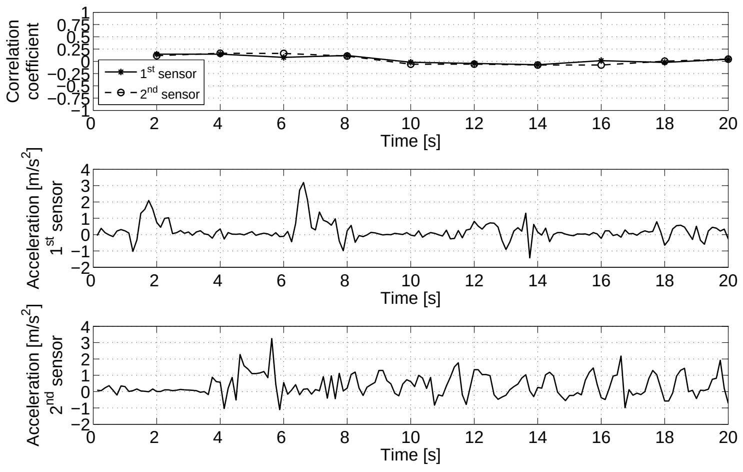
In our experiments, we are using the LIS3LV02DQ three-axis accelerometer from STMicroelectronics [24]. The price is around 15 USD and the typical power consumption is 2mW. The list of features include user selectable full scale of ±2g, ±6g, I^2C/SPI digital interface, programmable threshold for wake-up/free-fall and various sample rates up to 2.56kHz.

6.6.2 Experimental Results

Figure 6.5 shows the typical behaviour of the algorithm for joint and separate movements, over a period of 20 seconds. The plots at the top of the figures show the correlation coefficients calculated by the sensor nodes over the time history T, while the two bottom plots show the magnitude of the acceleration

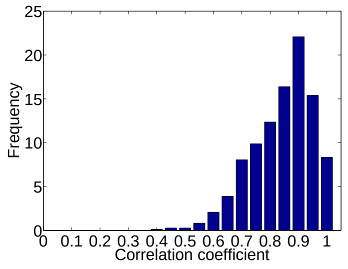


(a) Two nodes with accelerometers moving together.

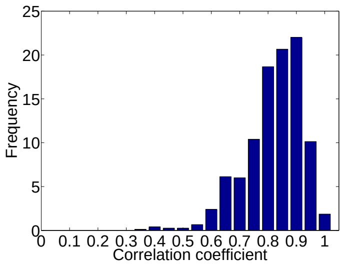


(b) Two nodes with accelerometers moving separately.

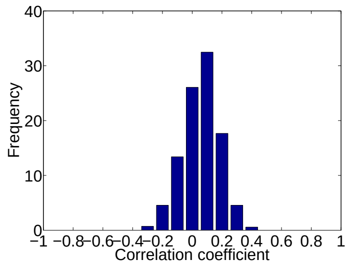
Figure 6.5: A typical behaviour of the algorithm for joint and separate movements, using accelerometers.



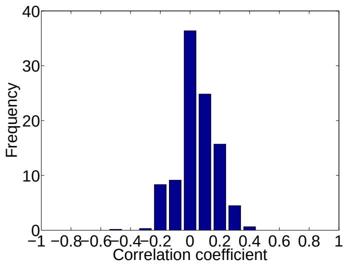
(a) Accelerometers on RTIs, moving together.



(b) Accelerometers in cars, moving together.



(c) Accelerometers on RTIs, moving separately.



(d) Accelerometers in cars, moving separately.

Figure 6.6: Histograms of correlation coefficients for accelerometers.

calculated by the sensors, relative to 1g (the constant gravitational component). We make the following observations:

- A node can deduce that it is static by calculating the standard deviation over the current data sequence k: a stationary situation can be assumed in case a relatively small standard deviation is recorded. The reason is that constant acceleration or speed cannot be achieved due to the vibrations registered by sensitive accelerometers. Therefore, similar to the tilt switch case, in static situations nodes may just send a short indication of their state.

- The method is successful in distinguishing between correlated and uncorrelated movements, for both types of experiments. A high correlation coefficient indicates that the sensor nodes move together (Figure 6.5(a)), while a low correlation coefficient shows a separate movement (Figure 6.5(b)).

We represent the histograms of the correlation coefficients in Figure 6.6, normalized to a percentage scale. We notice the difference between the correlation coefficients computed when sensors move together (Figures 6.6(a) and 6.6(b)) and separately (Figures 6.6(c) and 6.6(d)). A more detailed analysis of the results is given in Section 6.7.1.

6.7 Analysis

In this section, we discuss the two proposed solutions, and analyse the accuracy and scalability problems, pointing out the advantages and limitations.

6.7.1 Accuracy

One of the major questions is how accurate are the proposed methods. In Table 6.3, we present a brief statistical analysis of the results obtained from our experiments with RTIs and cars. The mean values of the correlation coefficient indicate a constant difference of more than 0.6 between joint and separate movement, in any of the listed settings. The standard deviation values suggest that the accelerometers provide more precise results. This fact is confirmed by the histograms from Figures 6.4 and 6.6. Knowing the correct output of the algorithm (joint or separate movement), the accuracy is computed as the percentage of correct decisions out of the total number of algorithm iterations. The decisions are based on a threshold $Th_C = 0.5$, which is the middle point between the value 0 for no correlation and 1, which corresponds to perfect correlation (for the computing a threshold which minimizes the sum of probabilities of an incorrect determination, see Chapter 7). The accelerometer-based solution proves more accurate, with 3.4% on average and a maximum of 5.2%. In addition, due to their better sensitivity, the accelerometers can identify reliably the separate movement situation.

A clustering algorithm that intends to reproduce the physical reality by grouping the nodes which are moving together can use this method for the detection of joint movement. The accuracy of the method has a great influence on the correctness of the clustering algorithm. To improve accuracy, (1) sensor fusion techniques can be used [96] and/or (2) decisions can be made over a larger time history, as shown within the next chapter.

Sensor	**Setting**	**Movement type**	**Mean**	**Stdev**	**Accuracy [%]**
Tilt switch	RTI	joint	0.641	0.087	95.89
Tilt switch	RTI	separate	-0.017	0.249	99.45
Tilt switch	car	joint	0.700	0.121	93.77
Tilt switch	car	separate	0.086	0.208	95.50
Accelerometer	RTI	joint	0.817	0.106	99.31
Accelerometer	RTI	separate	0.009	0.124	100
Accelerometer	car	joint	0.796	0.102	98.93
Accelerometer	car	separate	-0.003	0.127	100

Table 6.3: Statistical values.

6.7.2 Scalability

We present the factors that influence the maximum number of nodes supported by our proposed correlation methods. We denote the maximum number of neighbouring nodes as M. It follows that a node has maximum $M - 1$ neighbours, for which it computes the correlation coefficients.

Communication (Medium Access). We estimate the maximum number of neighbouring nodes M as follows. Each node transmits a data sequence every $k\Delta t$. If a TDMA-based MAC protocol is used, then the frame length T_f has to be at most $k\Delta t$, so that each node has a chance to transmit the data. The slot time T_s of a node is therefore bounded by $MT_s = T_f \leq k\Delta t$. Depending on the radio chip used, the slot time for sending a data packet can be computed. In our experiments, $T_s = 20$ms, which leads to $M = 100$.

Memory. The available memory (RAM and FLASH) is usually a critical resource on sensor nodes. The FLASH usage is not a problem, since the code memory footprint of our implementation on the sensor node platform amounts to 2.1kB out of 48kB available. Considering the recent low-power controllers equipped with 10kB RAM and computing the memory requirements for auxiliary sums and correlation data from Eq.6.6-6.8, we obtain a maximum number of nodes $M = 106$.

Execution Time. Since the correlation algorithm runs online, the nodes must have enough time within one slot to receive and process the incoming data. It follows that the execution time T_e must be smaller than the slot time: $T_e < T_s = T_f/M \Rightarrow M < T_f/T_e$. For the values used in our experiments, we get $M < 318$. This shows that the speed of the algorithm is not a limiting factor from the scalability point of view.

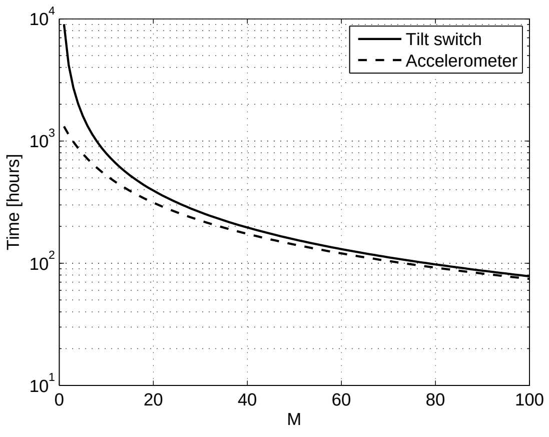


Figure 6.7: The running time for continuous movement.

Energy. Estimating the energy consumption is always important for the battery powered sensor nodes. We consider the radio communication and sensor operation as the most costly functions in terms of energy. For communicating the sampled data, a node performs $M-1$ receptions and one transmission every frame T_f. Typical radio current consumption on Ambient μNodes is 12.8mA for reception and 11mA for transmission. In addition, the current consumed with operating the sensors is 0.64μA for tilt switches and 0.65mA for accelerometers. Figure 6.7 shows the operating time for a node with a typical 1000mAh battery. The running time is represented depending on the number of neighbouring nodes M. The maximum values for M are deduced from the previous analysis, as being $M = 100$. As an example, for $M = 50$, the system can operate for approximatively 156 hours of continuous movement when using tilt switches and 142 hours when using accelerometers. However, the overall lifetime of a node increases if the movement intervals are short: during stationary periods, a node needs to send only an indication of its status, not the whole movement buffer.

6.7.3 Discussion

In what follows, we comment on the most important advantages and limitations of both solutions, giving also comparative details whenever relevant.

6.7.3.1 Advantages

1. *Autonomous joint movement recognition.* The proposed methods recognize the joint movement of vehicles on wheels, with the final aim at establishing groups autonomously. No infrastructure support is needed.

2. *Simplicity.* The overall system (hardware and software) is kept simple. This implies both a low price range and the feasibility of the implementation on resource constrained devices.

3. *Robustness to constructive differences of sensors.* The correlation coefficient gives an indication on the degree of similitude of two signals. It is known that the result is neither affected by scaling the signals with a certain factor, nor by adding/subtracting a constant value. This makes our method inherently robust to constructive differences of sensors, such as: calibration factors, zero-offset values, differences in sensitivity.

4. *Distinction between ensemble and separate movements.* Figures 6.4 and 6.6 indicate a good behaviour, with separable thresholds for distinguishing between ensemble and disjoint movement. However, the solution employing accelerometers proves more accurate in all tested situations.

5. *Implicit synchronization.* There is one important factor that can adversely affect the correctness of the correlation result, and that is time synchronization. It is therefore essential that the data sequences X_i, Y_i are synchronized when computing the correlation coefficient. For this reason, the incoming data from neighbours is correlated with the latest values sampled on the current node. Moreover, it is preferable not to use any retransmission mechanisms, since occasional packet losses do not affect the synchronization.

6. *Saving power while stationary.* In static situations, nodes can save energy by transmitting just a short indication of their status.

7. *Extended features.* More accurate results may be obtained by correlating extended movement features, such as direction or heading, speed, distance. In this sense, the accelerometer-based solution is much richer in possibilities.

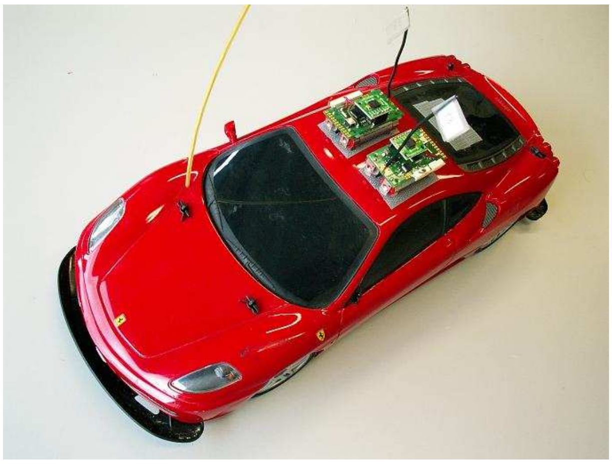


Figure 6.8: Two sensor nodes with accelerometers attached to a toy car.

6.7.3.2 Limitations

1. *Alignment and orientation.* Because movement is always relative to a frame of reference, different alignment or orientation of the sensors may produce misleading results. In the case of tilt switches, a similar alignment is necessary for obtaining a correct behaviour, such as in Figure 6.3(a). In contrast, for the accelerometer-based solution, no alignment is needed, since we are correlating the magnitude of the acceleration vector. This approach might also yield better results in the case of tilt switches, if a system such as Porcupine [104] is used, where each node is equipped with several switches oriented along different axis.

2. *Unpredictable delays.* Networking stack and sensor sampling delays can adversely affect the time synchronization of data sequences X_i, Y_i, eventually leading to errors in the correlation estimation. Timestamping the incoming Y_i at the receiver node $\mathbf{x}$, and choosing the corresponding X_i to correlate with, can alleviate the problem of networking stack delays.

3. *Placement on loose frames.* Throughout our tests, the sensor nodes are placed on the same rigid frame. We expect that the movement data becomes less correlated if the sensors are attached to loosely-coupled frames, such as the wagons of a train.

4. *Multihop networks.* Our solution is valid only for one-hop networks. A multihop network would impose a transitive correlation relation among sensor nodes. While the joint movement is a transitive relation, a low accuracy of the correlation algorithm can adversely affect the results for a multihop network. However, the algorithm could be improved in this case for example by computing the correlation coefficient using multiple connection paths.

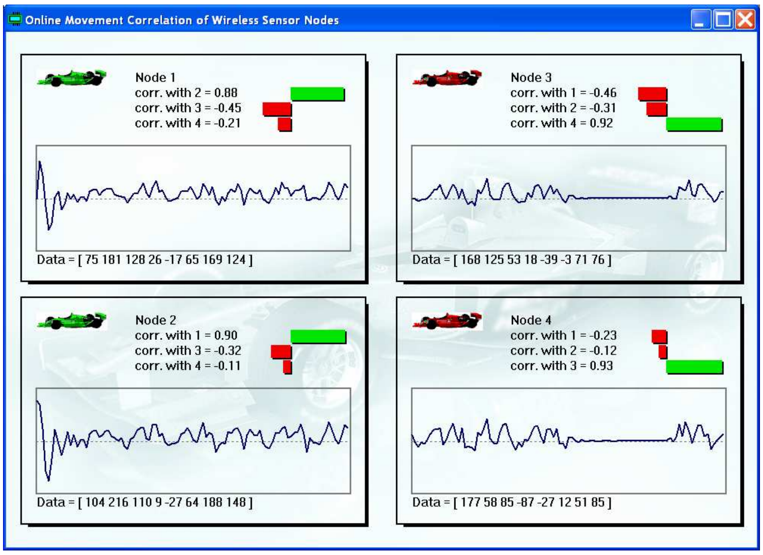
6.8 Demonstration

The demonstration [11] shows how the sensor nodes recognize online the joint movement. We use four sensor nodes with accelerometers, placed on two wirelessly controlled toy cars, as shown in Figure 6.8. The nodes convey their correlation decision by using different LED colours. In addition, a gateway node collects the correlation coefficients periodically broadcast by each moving node, together with the movement data, and logs them to the base station through a standard RS-232 interface. The base station provides a graphical user interface (see Figure 6.9), which shows the movement signals over a recent time history, the latest sampled data, the correlation coefficients between each two nodes and which nodes are considered together. Figure 6.9 displays a situation where the nodes 1 and 2 are together, being attached to the same car, while the nodes 3 and 4 are together on the other car controlled by a different remote controller.

Setting	**Correlation joint movement**	**Correlation separate movement**
RTIs	High	Low
Real cars	High	Low
Toy cars, different controllers	High	Low
Toy cars, same controller	High	High

Table 6.4: Behaviour of the algorithm for different experimental settings.

During the demonstration, we test the behaviour of the algorithm when the toy cars are also controlled by the same remote controller. The results show that when the cars move exactly in the same way, receiving the same commands from the controller, the nodes experience a high correlation coefficient, leading to the wrong conclusion that they are attached to the same car. In practice, however, we consider highly improbable that two vehicles move identically, fact shown



Node 1
corr. with 2 = 0.88
corr. with 3 = -0.45
corr. with 4 = -0.21
Data = [75 181 128 26 -17 65 169 124]

Node 3
corr. with 1 = -0.46
corr. with 2 = -0.31
corr. with 4 = 0.92
Data = [168 125 53 18 -39 -3 71 76]

Node 2
corr. with 1 = 0.90
corr. with 3 = -0.32
corr. with 4 = -0.11
Data = [104 216 110 9 -27 64 188 148]

Node 4
corr. with 1 = -0.23
corr. with 2 = -0.12
corr. with 3 = 0.93
Data = [177 58 85 -87 -27 12 51 85]

Figure 6.9: Demonstration interface.

by the experiments with RTIs and cars. Table 6.4 summarizes the behaviour of the algorithm under various experimental settings.

6.9 Conclusions

This chapter proposes a method for recognizing joint and separate movement of wireless sensor nodes. Nodes are moving together if their movements correlate for a certain amount of time. For extracting the movement information, we investigate two solutions, one using tilt switches, the other one using accelerometers. On the one hand, the solution using tilt switches proves to be cheaper, simpler and less energy consuming. On the other hand, the solution using accelerometers is more reliable in distinguishing between ensemble and separate movements and it does not need any sensor alignment. Nevertheless, the solution is more complex, as the magnitude of the acceleration has to be calculated from the three samples corresponding to the three axes. The scalability analysis shows a maximal network density of 100 nodes for both solutions.

Extracting semantic information about sensor nodes is essential to achieve context-aware clustering. In the next chapter, we propose a clustering mechanism that uses the semantic information in the decision process, such that nodes with similar semantic properties are grouped together with the final goal of providing a service.

Appendix

The proofs of Eq. 6.6 and 6.7 are the following:

$$\begin{aligned}
var_i(X) &= \sum_{j=(i-n)k+1}^{ik} \frac{(x_j-\bar{X}_i)^2}{N} = \sum_{j=(i-n)k+1}^{ik} \frac{x_j^2}{N} - \bar{X}_i^2 = \\
&= (\sum_{j=(i-n-1)k+1}^{(i-1)k} \frac{x_j^2}{N} - \bar{X}_{i-1}^2) + \sum_{j=(i-1)k+1}^{ik} \frac{x_j^2}{N} - \\
&- \sum_{j=(i-n+1)k+1}^{(i-n)k} \frac{x_j^2}{N} - \bar{X}_i^2 + \bar{X}_{i-1}^2 = \\
&= var_{i-1}(X) + \frac{\sigma_i^x - \sigma_{i-n}^x}{N} - (\bar{X}_i^2 - \bar{X}_{i-1}^2)
\end{aligned}$$

$$\begin{aligned}
cov_i(X,Y) &= \sum_{j=(i-n)k+1}^{ik} \frac{(x_j-\bar{X}_i)(y_j-\bar{Y}_i)}{N} = \sum_{j=(i-n)k+1}^{ik} \frac{x_j y_j}{N} - \bar{X}_i\bar{Y}_i = \\
&= (\sum_{j=(i-n-1)k+1}^{(i-1)k} \frac{x_j y_j}{N} - \bar{X}_{i-1}\bar{Y}_{i-1}) + \sum_{j=(i-1)k+1}^{ik} \frac{x_j y_j}{N} - \\
&- \sum_{j=(i-n-1)k+1}^{(i-n)k} \frac{x_j y_j}{N} - \bar{X}_i\bar{Y}_i + \bar{X}_{i-1}\bar{Y}_{i-1} = \\
&= cov_{i-1}(X,Y) + \frac{S_i^{xy} - S_{i-n}^{xy}}{N} - (\bar{X}_i\bar{Y}_i - \bar{X}_{i-1}\bar{Y}_{i-1})
\end{aligned}$$

The size of binary data for integer computation is the following:

Variable	**Size**	**Representation**
k	2^3	int8
N	2^7	int8
$x_i,\ y_i$	2^8	int8
$S_i^x,\ S_i^y$	2^{11}	int16
$\sigma_i^x,\ \sigma_i^y,\ S_i^{xy}$	2^{20}	int32
$N\bar{X}_i,\ N\bar{Y}_i$	2^{15}	int16
$N^2\bar{X}_i^2,\ N^2\bar{Y}_i^2,\ N^2\bar{X}_i\bar{Y}_i$	2^{30}	int32
$N(\sigma_i^x),\ N(\sigma_i^y),\ NS_i^{xy}$	2^{26}	int32
$N^2var_i(X),\ N^2var_i(Y),\ N^2cov_i(X,Y)$	2^{32}	int32

Table 6.5: Binary data size.

Chapter 7

A context-aware method for spontaneous clustering of wireless sensor nodes

Wireless sensor nodes attached to everyday objects and worn by people are able to collaborate and assist users in their activities. We propose a method through which wireless sensor nodes organize spontaneously into clusters based on a common context. Given that the confidence of sharing a common context varies in time, the algorithm takes into account a window-based history of contextual information. We approximate the behaviour of the algorithm using a Markov chain model and we analyse theoretically the cluster stability. We compare the theoretical approximation with simulations, by making use of experimental results from field tests. We show the tradeoff between the length of the time history necessary to achieve a certain stability and the responsiveness of the clustering algorithm.

7.1 Introduction

Wireless sensor networks, smart everyday objects and cooperative artefacts represent all different facets of the ubiquitous computing vision, where sensor-enabled devices are integrated in the environment and provide context-aware services to the users. Various systems have already shown how to retrieve the context, such as the physical context (e.g. position, movement [80]), the situation (e.g. meeting [161]) and even the emotional context (e.g. mood detection [78]). One step further is to have sensor nodes that interact and use common contextual information for reasoning and taking decisions locally. Such a "networked world" opens perspectives for novel applications in numerous fields, including transport and logistics (see Chapter 1), industrial manufacturing [119], healthcare [115], civil security and disaster management [122].

The Smart-Its project [89] introduces the notion of context sharing: two smart objects are associated by shaking them together. Using this explicit interaction between the two devices, an application-level connection can be established. For example, the two devices can authenticate using secret keys that are generated based on the movement data [123].

In this chapter, we explore a non-traditional networking paradigm based on context sharing. Previous work shows that sensor nodes can recognize a common context and build associations of the type "moving together". Firstly, Lester et al. [109] use accelerometer data to determine if two devices are carried by the same person. The authors use a coherence function to derive whether the two signals are correlated at a particular frequency. Secondly, in Chapter 6, we propose a correlation algorithm which determines whether dynamic sensor nodes attached to vehicles on wheels move together. Starting from this results, we address the problem of organizing the nodes sharing a common context into stable clusters, given the dynamics of the network and the accuracy of the context-recognition algorithm.

The contributions of this chapter are as follows. Firstly, we propose Tandem, an algorithm for spontaneous clustering of mobile wireless sensor nodes. The algorithm allows reclustering in case of topological or contextual changes, and achieves stable clusters if there are no changes in the network, by analysing the similarity of the context over a time history. Secondly, we approximate the behaviour of the algorithm using a Markov chain model, which allows us to estimate the percentage of time the clustering structure is correct given that the topology is stable. Thus, we are able to analyse the cluster stability both theoretically and through simulations, using experimental results from real field tests. Thirdly, we study the tradeoff between the time history necessary to achieve a certain stability and the responsiveness of the clustering algorihm. As a result, we estimate the delay induced by the time history, given a desired cluster stability.

7.2 Application scenarios

We describe two applications where wireless sensor nodes are able to extract and communicate the general contextual information for creating a dynamic cluster. The clusters then provide services such as reporting the group membership, analysing the cluster activity, recognizing fine-grained events and actions.

7.2.1 Transport and logistics

Using wireless sensor networks technology in transport and logistics is particularly interesting for dynamically locating the goods, generating automatic packing lists, as well as for monitoring the storage condition of a product (e.g. temperature, light) or its surroundings (see Chapter 1). Figure 7.1 shows the general idea of context-aware grouping in the transport and logistics scenario, where products loaded into rolling containers correlate their movement information and report as a group to the transport company personnel.

The list generator for automatic packing described by Antifakos et al. works in a similar manner [33]. Various order items are packed in a box and an invoice is generated. In order to find out where a certain item is, nodes with movement sensors can be attached to each good. When the box is moved around, the items inside can correlate their movement and decide that they form a group. When the objects are grouped, checking on the goods and packing lists can be done automatically.

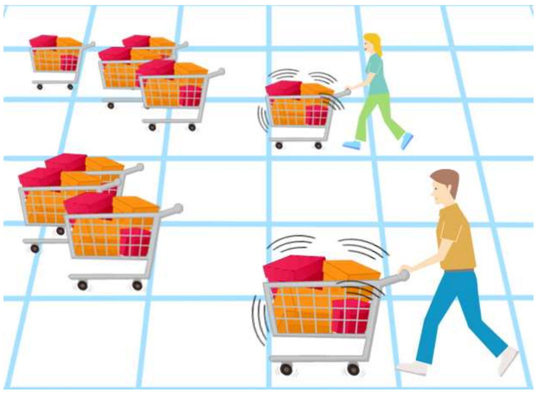


Figure 7.1: Transport and logistics scenario.

7.2.2 Body area networks

Wearable computing aims at supporting workers or people in everyday life by delivering context-aware services. One important aspect is the recognition of human activities, which can be inferred from sensors integrated into garments and objects people are interacting with, denoted by Body Area Networks (BANs). The usage of context information enables a more natural interaction between humans and computers, and can assist workers with complex tasks with just now relevant information. Examples include training unskilled workers for assembly tasks [146], monitoring the health and activity of patients [115], assisting firefighters engaged in rescue operations in unknown environments with poor visibility [70].

Clustering of nodes based on the movement of persons simplifies the selection of relevant sensors from the environment (i.e. only the nodes attached to or handled by the person) which can contribute to the recognition of the currently performed activity. The advantage of clustering is that the recognition processing can be kept within the cluster, which is important for environments where multiple people are present. Therefore, clustering can be used to provide: (1) identification of the body wearing the sensors, (2) a better recognition stability when selecting sensors moving with a person for the recognition task, and (3) potentially a trusted network where private data can be communicated only to nodes within the cluster.

7.3 Algorithm description

The goal of Tandem is to organize the nodes that share the same context, so that they can subsequently collaborate to provide a service. Tandem makes the following assumptions:

Assumption 6 *Each node periodically runs a shared-context recognition algorithm with its neighbours. This algorithm provides a number on a scale, representing the confidence value that two nodes are together.*

The confidence value can be for example the output of a *coherence* function, which measures the extent to which two signals are linearly related at each frequency, on the scale from 0 to 1 [109], or the *correlation coefficient*, which indicates the strength and direction of a linear relationship, on the scale from -1 to 1 (see Chapter 6). The shared-context recognition algorithm permanently evaluates the context, so that at each time step every node has an updated image of the current situation, reflected in a new set of confidence values (one for every neighbour). Since the context recognition algorithm has a certain accuracy, the perceived context determined by the confidence values may vary in time.

7.3.1 Requirements

Following the *Transport and logistics* and *Body area networks* scenarios from Section 7.2, the nodes sharing the same context are within each-others transmission range, so we consider only one-hop clusters. The environment is dynamic, where the topology and context can change at any time. The requirements for the clustering algorithm are the following:

1. *Incorporate dynamics.* The clusters can merge or split, depending on the context changes. Nodes can join and leave the cluster if the topology or context changes. For example, in the BAN scenario, people can pick up and use different tools, and then return or exchange them with other people. In this case, the nodes attached to the tools have to join and leave the BAN clusters. Contextual and topological changes cannot be predicted, so the algorithm cannot assume a stable situation during cluster formation.

2. *Stability.* If there are no contextual or topological changes, the clustering structure has to be stable. Following the remark that every node periodically re-evaluates the shared context with its neighbours, the fluctuations of the confidence values may lead to unwanted changes in the cluster structure. Therefore, the cluster has to cope with these fluctuations in order to keep the structure as stable as possible. A solution to increase the stability is to analyse the similarity of the context over a larger time interval. In this sense, a tradeoff has to be found between the spontaneity in accommodating changes and the desired cluster stability.

3. *Energy-efficiency.* The communication overhead should be kept to a minimum, for prolonging the lifetime of the wireless network.

4. *Facilitate service provisioning.* The clusters have to interact with the higher-layer applications and provide context-aware services to the user.

Algorithm 5: Tandem - node v (events/actions)

```
Initialization:
 1. $r(v) \leftarrow \perp,\ r(u) \leftarrow \perp,\ \forall u \in \Gamma(v)$

SelectParent:
 1. $r_0(v) \leftarrow r(v)$, $\Gamma_0(v) \leftarrow \Gamma(v)$
 2. Update $\Gamma(v)$, $N_1(v)$, $N_2(v)$, $y(v)$          // $N_1 \leftarrow \{m \in \Gamma(v) \mid h(v,m) > h_{min}\}$
 3. if $N_1 = \emptyset$ then
 4.    $r(v) \leftarrow \perp$          // Become unassigned
 5. else
 6.    if $N_2(v) \neq \emptyset$ then
 7.       if $(r(v) = \perp) \vee (r(v) \neq v \wedge r(v) \notin N_2(v)) \vee (r(v) = v \wedge pn(v) < pn(y(v)))$ then
 8.          $r(v) \leftarrow y(v)$          // Choose $y(v)$ as the root of $v$
 9.       end if
10.    else if $r(v) \neq v$ then
11.       if $\{\ u \in N_1 \mid r(u) \neq \perp\} = \emptyset$ then
12.          $r(v) \leftarrow v$          // Become root
13.       else
14.          $r(v) \leftarrow \perp$
15.       end if
16.    end if
17. end if
18. if $(r(v) \neq r_0) \vee (\Gamma(v) \setminus \Gamma_0(v) \neq \emptyset)$ then
19.    Send SetRoot $(v, r(v))$ to neighbours          // Announce root change
20. end if

SetRoot(u,r): // Update the information from neighbour u
 1. $r(u) \leftarrow r$
```

7.3.2 Cluster formation algorithm

Tandem represents a simplification of the generalized clustering algorithm described in Chapter 4, where for every node v, the generic sets $N_1(v), N_2(v)$ and conditions $P_1(v), P_2(v), P_3(v)$ are replaced with the specific instances of Tandem. Each node v periodically evaluates the confidence of sharing the same context with its neighbours. If the confidence with a neighbour u exceeds a certain threshold, then v considers that it shares the same context with u for the given time step. The final decision for sharing the same context with u is founded on the confidence values from a number of previous time steps, called the *time history* (see Section 7.4.1).

To provide a consistent management of the membership list and cluster organization, a *clusterhead* or *root* node is dynamically elected among the nodes that share the same context. This also assists the service provisioning, by having a single point of interaction with the higher-layer applications (see Requirement 4 from Section 7.3.1). In order to allow merging of clusters and to facilitate the election process, each node is assigned a unique *priority number*, either based on the node's hardware address, the resources available or as a context-dependant measure. A regular node subscribes to the clusterhead with which it shares a common context and which has the highest priority number.

We use the following notation, in addition to the notation defined in Chapter 4:

- $pn(v)$ is the priority number of node v, corresponding to the weight $w(v)$.
- $h(v, u)$ represents the number of times v and u share a common context over a total time history of H steps.
- h_{min} is the minimum number of times when two nodes are sharing a common context, such that they can safely be considered part of the same cluster; we redefine the notation from Chapter 4: if $h(v, u) > h_{min}$ then $s(v, u) = 1$, else $s(v, u) = 0$.

Tandem constructs a set of one-hop clusters, based on the context information shared by the nodes. A node v can be: (1) *unassigned*, where v is not part of any cluster, (2) *root*, where v is clusterhead, or (3) *assigned*, where v is assigned to a cluster where the root node is one of its neighbours.

Algorithm 5 gives the detailed description of the cluster formation and update of knowledge among neighbouring nodes. Every node has the following information about its neighbours: the root, the priority number and whether it shares a common context for a specified time history. Let v be an arbitrary node in the network. At each time step, node v changes or chooses its root node in the following cases: (1) v is unassigned, (2) v does not share a common context with its root, (3) the root of v is no longer a root or (4) v is root and there is another neighbour root, sharing the same context with v, that has a higher priority number. In any of these cases, v chooses as root node the neighbour root u with which it shares a common context and which has the highest priority number. If such a neighbour does not exist, v competes for clusterhead or becomes unassigned. The decision is based on the current status of the neighbours and tries to minimize the effect of the following erroneous situation: due to context fluctuations, an assigned node v may loose its root node and cannot join another cluster because none of its neighbours is a root. Therefore, v may become root, form a new cluster and attract other nodes in that cluster. To avoid this undesirable outcome, a node declares itself root only if all its neighbours with which it shares a common context are unassigned. If there exists at least one neighbour u with which v shares a common context and u has a valid root node, then v becomes unassigned.

Node v announces the changes in choosing the root node by sending a local broadcast message *SetRoot* to its neighbours. In case of topological changes, this message is also used to announce the new neighbours of the current structure. Tandem thus allows cluster merging or splitting, which meets the Requirement 1 from Section 7.3.1. We make the observation that there is no additional strain in terms of communication overhead on the clusterheads compared to the other nodes (see Algorithm 5).

Let us consider the example from Figure 7.2. A BAN is associated with each person, consisting of five nodes placed in various locations: backpack, belt, pocket and wrist. The clustering structure is seen from the perspective of node 4, which is attached to the wrist of the left person. The clusterheads are represented with big circles (nodes 1 and 8). The dark-coloured arrows indicate the assignment of the other nodes to the current clusterheads. Node 7 is unassigned, as the shared-context recognition algorithm did not associate this node with any of the neighbouring clusterheads at the previous time step. The light-coloured arrows show the confidence values of node 4 at the current time step, using the coherence function. The confidence values for the nodes on the same body with node 4 range between 0.66 and 0.88, while for the other body they lie between 0.39 and 0.57. Because the confidence of sharing the same context with the root node 1 is 0.66 and above the threshold of 0.6, node 4 keeps the current root (see Section 7.4.1 for the computation of the threshold value). Otherwise, it would have become unassigned (node 4 has some neighbours with the same context, having a valid root node), or assigned to the other cluster, if the confidence value for the neighbouring root node 8 was higher than the threshold.

7.4 Cluster stability analysis

Several algorithms for context sharing have been proposed in the literature, using various sensors and providing different accuracies (see Section 7.1). However, none of them gives a measure of the overall accuracy of the system (i.e. how well does it mimic the reality), when multiple nodes sharing different contexts come together. We would like to analyse the cluster stability from both the theoretical point of view, by giving the average, upper and lower bounds,

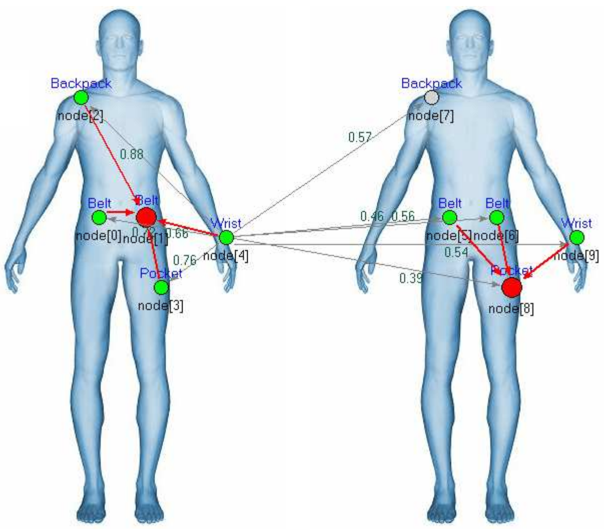


Figure 7.2: Graphical simulation of the clustering algorithm on BANs.

and through simulations. In addition, we are interested in the tradeoff between the time history necessary to achieve a certain stability and the responsiveness of the clustering algorithm. First, we compute the probabilities of correctly assessing the context, given the distribution of the confidence values. Second, we model the output of the algorithm using Markov chains and we derive an approximation for the proportion of time the clustering structure is in a correct state.

7.4.1 Determination of common context

In this section, we give an example of how the probabilities of correct detection of the shared context can be computed.

Let v be a node in the network and u a neighbour of v. If v does not share the same context with u (e.g. they represent sensor nodes attached to different persons), we model the confidence value computed by the shared-context recognition algorithm with the random variable $X_1(v,u)$. If v shares the same context with u (e.g. they are attached to the same person), we model the confidence value as a random variable $X_2(v,u)$. We take the distribution encountered during the experiments as the reference Probability Density Function (PDF): we associate the random variables $X_1(v,u)$ with the PDF φ_1 and the corresponding

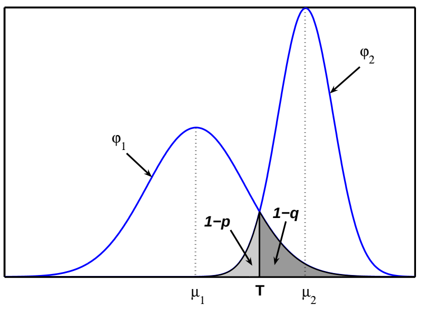


Figure 7.3: The calculation of the threshold value T and the probabilities p (correct detection of a common context) and q (correct detection of different contexts).

Cumulative Distribution Function (CDF) Φ_1. Similarly, we associate the random variables $X_2(v, u)$ with the PDF φ_2 and CDF Φ_2. We make the following assumptions:

Assumption 7 *$X_1(v, u)$ and $X_2(v, u)$ are normally distributed.*

Assumption 8 *φ_1 and φ_2 are the same for every pair of nodes (v, u).*

Node v selects the subset of its neighbours with which it shares a common context based on a threshold value T. We choose T as the intersection point of the two PDFs φ_1 and φ_2, since this minimizes the sum of probabilities of an incorrect determination. We denote p as the probability of a correct detection of a *common* context and q as the probability of a correct detection of *different* contexts. The probabilities p and q are computed in the following way (see Figure 7.3):

$$p = 1 - \Phi_2(T), \qquad q = \Phi_1(T) \tag{7.1}$$

We compute the threshold value for the case where the distributions are normal, which is valid for the applications described in Section 7.2 (see the experimental data from Chapter 6 and [109]). Let us consider two normal distributions, $\varphi_1(\mu_1, \sigma_1)$ and $\varphi_2(\mu_2, \sigma_2)$, where $\mu_1 \neq \mu_2$. The intersection point of φ_1 and φ_2 which lies between μ_1 and μ_2 is the following:

$$\begin{cases} T = \frac{\mu_1+\mu_2}{2} & \text{for} \quad \sigma_1 = \sigma_2 \\ T = \frac{\mu_1\sigma_2^2-\mu_2\sigma_1^2+\sigma_1\sigma_2\sqrt{(\mu_1-\mu_2)^2+2(\sigma_2^2-\sigma_1^2)ln(\sigma_2/\sigma_1)}}{\sigma_2^2-\sigma_1^2} & \text{for} \quad \sigma_1 \neq \sigma_2 \end{cases} \tag{7.2}$$

Using Eq. 7.1 and 7.2, it is straightforward to compute p and q, knowing the characteristics of φ_1 and φ_2. We are now interested in how these probabilities change if we involve the time history in the decision process. The probability p_h of the correct detection that two nodes share a common context for a minimum time history h_{min} out of a total of H time steps is given by the CDF of the binomial distribution:

$$p_h(h_{min}, H) = \sum_{k=h_{min}}^{H} \left(\begin{array}{c} H \\ k \end{array} \right) p^k(1-p)^{H-k} \tag{7.3}$$

Similarly, the probability q_h of the correct detection of different contexts for a minimum time history h_{min} out of a total of H time steps is:

$$q_h(h_{min}, H) = \sum_{k=h_{min}}^{H} \left(\begin{array}{c} H \\ k \end{array} \right) q^k(1-q)^{H-k} \tag{7.4}$$

We have therefore $p = p_h(1,1)$ and $q = q_h(1,1)$.

Using these probabilities, we model the output of the algorithm using Markov chains, as described within the next section.

7.4.2 Modelling with Markov chains

We approximate the behaviour of the algorithm with a Markov chain, which allows us to estimate the global probability of having a correct cluster. We stress on the difference between a *time step* and a *Markov chain step.* A time step is related to the periodic update of the context information by the shared-context recognition algorithm which runs on every node. For improving the probabilities of a correct detection of a shared context, the algorithm looks over a time history H, composed of a number of time steps (see Section 7.4.1). A Markov chain step is the "memoryless" transition from one state to another, which happens on intervals equal to the total time history H.

We define a *group* G as the collection of nodes that share the same context in reality. We define a *cluster* C as the collection of nodes which have the same root node (as a result of Agorithm 1). The goal of the clustering algorithm is that for any group of nodes G, there exists a cluster with the root $r_0 \in G$ such that $\forall v \in V, r(v) = r_0 \Leftrightarrow v \in G$. Taking the example from Figure 7.2, we have two groups: $G_1 = \{0, 1, 2, 3, 4\}$, $G_2 = \{5, 6, 7, 8, 9\}$ and two clusters: $C_1 = \{0, 1, 2, 3, 4\}$ with root node 1, $C_2 = \{5, 6, 8, 9\}$ with root node 8. Node 7 is unassigned, and thus it is not part of any cluster.

We define the following states for a cluster C:

1. *Correct:* The cluster has a root node from the group and all the members of the group are part of the cluster (we intentionally take $G \subseteq C$; the nodes from $C \setminus G$ are part of other groups, which have the corresponding clusters in an incorrect state).

2. *Has root:* The cluster has a root, but not all the members of the group are part of the cluster.

3. *No root:* None of the cluster members is root.

4. *Election:* After reaching state 3, members of the cluster start an election process for choosing the root node.

For example, cluster C_1 from Figure 7.2 is *Correct*, while C_2 is in the state *Has root*, since node 7 is unassigned.

The Markov chain determined by the transition matrix P:

$$P = \begin{pmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \\ p_{41} & p_{42} & p_{43} & p_{44} \end{pmatrix}$$

Let $m \geq 0$ be the number of root nodes with higher priority than the current root and $k \geq 1$ the number of nodes in the cluster. If the cluster has a root, let r_0 be the root node. The probabilities p_{ij} are evaluated in a worst case scenario, by minimizing the chance to get in the *Correct* state.

The conditions which determine the probabilities p_{ij} are the following:

- p_{11}: (a) r_0 remains root in the next step, as r_0 does not share the same context with other root nodes with higher priority (i.e. q^m), and (b) all other nodes in the group share the same context with r_0 (i.e. p^{k-1}).

- p_{12}: (a) r_0 remains root in the next step (i.e. q^m), and (b) there exists at least one node in the group that does not share the same context with r_0 (i.e. $1 - p^{k-1}$).

Probability	**Value**	**Probability**	**Value**
p_{11}	$q^m p^{k-1}$	p_{21}	$q^m p^{k-1} q^{m(k-1)}$
p_{12}	$q^m(1 - p^{k-1})$	p_{22}	$q^m(1 - p^{k-1} q^{m(k-1)})$
p_{13}	$1 - q^m$	p_{23}	$1 - q^m$
p_{14}	0	p_{24}	0
p_{31}	0	p_{41}	0
p_{32}	0	p_{42}	$q^{mk}(1 - (1-p)^{k(k-1)})$
p_{33}	0	p_{43}	0
p_{34}	1	p_{44}	$1 - q^{mk}(1 - (1-p)^{k(k-1)})$

Table 7.1: Transition probabilities p_{ij}, where p is the probability of a correct detection of a *common* context, q is the probability of a correct detection of *different* contexts, m is number of root nodes with higher priority than the current root and k is the number of nodes in the cluster.

- p_{13}: r_0 shares the same context with a root node with higher priority, so that it gives up its role and joins another cluster (i.e. $1 - q^m$).

- p_{21}: (a) r_0 remains root in the next step (i.e. q^m), (b) all other the nodes in the group do not share the same context with other root nodes with higher priority (i.e. $q^{m(k-1)}$) and (c) all other nodes in the group share the same context with r_0 (i.e. p^{k-1}).

- p_{23}: r_0 shares the same context with a root node with higher priority, so that it gives up its role and joins another cluster (i.e. $1 - q^m$).

- p_{34}: from state *No root* the system goes at the next step to state *Election*.

- p_{42}: (a) all the nodes in the group do not share the same context with any root node with higher priority (i.e. q^{mk}), and (b) there are at least two nodes in the group that share the same context (i.e. $1 - (1-p)^{k(k-1)}$).

Table 7.4.2 gives the computed probabilities for each transition of the Markov chain. We notice that the *Correct* state can be reached only from the *Has root* state. If the number of root nodes with higher priority than the current root is greater than 0 (i.e. $m > 0$), the probability p_{21} is minimized, so that in the stationary distribution of the Markov chain, the probability to be in the *Correct* state is lower than the real probability. Calculating the stationary distribution of the Markov chain yields the following result [79]:

$$p_1(m,k,q,p) = \frac{p_{21}p_{42}}{(1+p_{21}-p_{11})(p_{42}+p_{42}p_{13}+p_{13})} \tag{7.5}$$

We define the *cluster stability* P_S as the probability of a cluster to be in the *Correct* state. The most stable cluster is the one having the root with the highest priority number. The least stable cluster is the one having the root with the lowest priority number. Given that there are c clusters in the network, we have therefore the following lower and upper bounds for the cluster stability:

$$p_1(c-1,k,q,p) \leq P_S \leq p_1(0,k,q,p) \tag{7.6}$$

P_S can be approximated with the stability of the cluster with a lower priority than other $\frac{c-1}{2}$ clusters:

$$P_S \approx p_1(\frac{c-1}{2},k,q,p) \tag{7.7}$$

Using these theoretical estimations, we can determine the time history necessary to achieve a certain cluster stability, as shown within the next section.

7.5 Results

We analyse the performance of Tandem by running a series of simulations in the OMNeT++ [157] simulation environment. As the cluster formation and stability is affected only by the nodes in the one-hop neighbourhood, we simulate a network where the nodes are attached to different mobile objects or people, and where there is a wireless link between any two nodes. We focus on a mobile scenario with clustered nodes moving around and passing each other, and we analyse how the cluster stability changes when we vary the number of groups. First, we compare Tandem with a traditional clustering method and point out the factors that determine the improved behaviour of our algorithm. Second, based on the experimental results from Lester et al. [109] and Chapter 6, we evaluate Tandem with respect to cluster stability and communication overhead.

7.5.1 Comparison to traditional clustering

Tandem groups the nodes with similar semantic properties, namely the shared context. The uncertainty in determining the shared context and the resulting variation in time may lead to instability of the clustering structure. Therefore, an algorithm that generates stable clusters distinguishes itself from generic algorithms designed for ad-hoc networks by paying attention especially to minimizing the effect of the variation of confidence values.

To illustrate this difference, we present an alternative clustering algorithm following the idea of DMAC [43], denoted by DMAC*. In DMAC*, each node v takes into account every change in the confidence value. DMAC* can be seen as a particular case of Algorithm 1 from Chapter 4, where the definitions of the sets $N_1(v), N_2(v)$ and conditions $P_1(v), P_2(v), P_3(v)$ are the following:

- **DMAC***:
  - $N_1 = \{m \in \Gamma(v) \mid h(v,m) > h_{min}\}$
  - $N_2(v) = \{u \in N_1(v) \mid w(u) > w(v) \wedge r(u) = u\}$
  - $P_1(v),\ P_2(v) : true$
  - $P_3(v) : false$

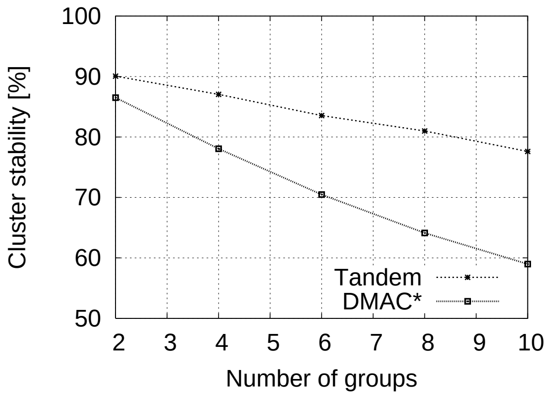
We stress on the following two main differences of DMAC* compared to Tandem:

1. From the definitions of N_1 and P_1, it follows that if v detects in its neighbourhood a clusterhead r_1 with a higher priority value than the current clusterhead r_0, and v shares a common context with r_1, then v directly chooses r_1 as clusterhead.

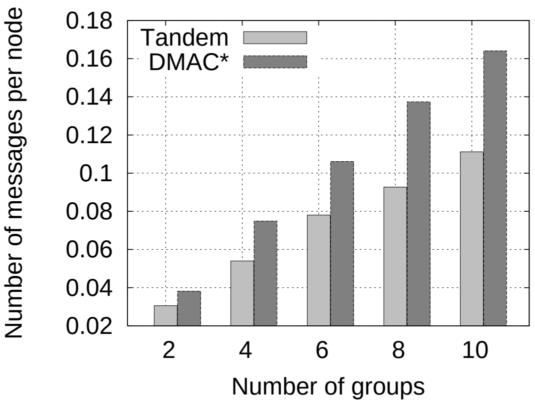
2. From the definition of P_2, we have that if none of its neighbours sharing a common context is clusterhead, then v declares itself a clusterhead.

The first condition may lead to unwanted effects, where a single error in the detection of different contexts between v and r_1 changes the cluster membership of v. Likewise, an error in the detection of the common context between v and r_0 results through the second condition into the erroneous situation of having two clusterheads for the same group.

For evaluating the performance of Tandem compared to DMAC*, we run a series of simulations with groups of 10 nodes. We vary the number of groups between 2 and 10, and the probabilities p and q between 0.9 and 0.99. We measure the cluster stability in percentage (see Section 7.4.2) and the energy efficiency as a function of the number of *SetRoot* messages sent by every node in one time step. Due to the two conditions mentioned above, Tandem outperforms DMAC* in all situations considered. As an example, we show in Figure 7.4 the average simulation results for $p = q = 0.99$ and various number of groups. The 5^{th} and 95^{th} percentiles are presented in the Appendix. We notice that Tandem improves the stability up to 18% on an absolute scale and the energy efficiency up to 32% on a relative scale, when compared with DMAC*. Therefore, Tandem is more stable and energy-efficient than DMAC*, and the difference grows when increasing the number of groups. In this way, Requirements 2 and 3 from Section 7.3.1 are satisfied.



(a) Comparison of cluster stability.



(b) Comparison of energy efficiency.

Figure 7.4: Comparison between Tandem and DMAC*.

7.5.2 Evaluation

A typical example of context sharing is the similarity of movement, which we analyse in this section using experimental results corresponding to the *Transport and logistics* and *Body area networks* scenarios described in Section 7.2. In general, the movement information is extracted from accelerometers. Simpler sensors such as tilt switches can be also used, but with less accurate results. We have the following two concrete examples of wireless objects moving together:

1. **RTI** - wireless sensor nodes used in the *Transport and logistics* scenario, which correlate their movement pattern; the shared-context recognition algorithm computes a correlation coefficient between the movement data of two neighbouring nodes, which is extracted using both tilt switches and accelerometers (see Chapter 6).

2. **BAN** - smart devices in the *Body area networks* scenario, which decide whether they are carried by the same person; the shared-context recognition algorithm uses a coherence function of the movement data provided by accelerometers [109].

Table 7.2 shows the characteristics of the normal distributions derived from the concrete experiments conducted in both application examples, together with the computed threshold from Eq. 7.2 and the probabilities p and q. Contrary to the RTI scenario, where the nodes moving together experience exactly the same movement pattern, in the BAN scenario different parts of the body are engaged in different types of movements during walking. For a realistic evaluation, we choose the worse case experimental results from the BAN scenario, where the nodes are attached to the pocket and wrist of the subjects (Pocket/Wrist trial [109]). As mentioned earlier, the RTI scenario uses the correlation coefficient for recognizing the common context, which takes values in the interval

Application	μ_1	σ_1	μ_2	σ_2	T	p	q
RTI - tilt switch	-0.017	0.249	0.641	0.087	0.438	0.9902	0.9662
RTI - accelerometer	0.009	0.124	0.817	0.106	0.442	0.9998	0.9998
BAN - Pocket/Wrist	0.519	0.069	0.757	0.065	0.640	0.9636	0.9607

Table 7.2: Statistics from the experiments (mean and standard deviation of confidence values for separate (μ_1, σ_1) and joint (μ_2, σ_2) movements), the threshold value and computed probabilities of correct detection of common (p) and different (q) contexts.

$[-1, 1]$, while the BAN scenario uses the output of a coherence function in the range of $[0, 1]$. Therefore, the threshold for the BAN scenario is higher than the thresholds for the RTI scenario (see Table 7.2).

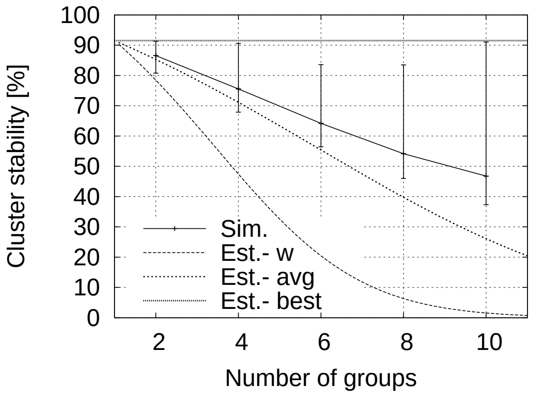
The scalability analysis of the movement correlation method proposed for the RTI scenario indicates a maximum network density of 100 nodes (see Chapter 6), imposed by the shared wireless medium. Because the same periodic transmission of the movement data is needed for the BAN scenario [109], we use in our simulations the same maximum network density for both applications. We have 10 nodes in each group moving together and interacting with other groups, and we vary the number of groups between 2 and 10 and also the time history. We recall from Section 7.4.2 that the cluster stability is the probability that the cluster is in the *Correct* state. We represent the cluster stability in percentage, namely (1) the average simulation results, (2) the estimation of the worst case, derived from Eq. 7.6, (3) the estimation of the average case, derived from Eq. 7.7, and (4) the estimation of the best case, derived from Eq. 7.6.

For each point on the simulation plots we run up to 10 simulations of $10^4 - 10^5$ time steps. In order to study the influence of the history size, we take as the output of the algorithm the majority over the time history. For that, we have $H = 2h_{min} - 1$ and we vary h_{min} from 1 to 4.

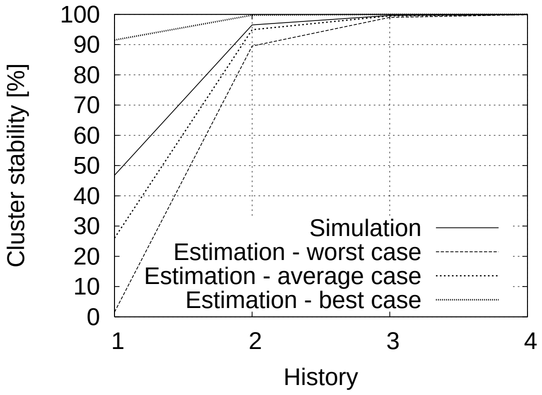
7.5.2.1 RTI with tilt switches scenario.

Figure 7.5(a) shows the cluster stability depending on the number of groups present in the network, given that the time history is $h_{min} = 1$. The error bars represent the absolute minimum and maximum stability recorded during the simulations. We notice that the results respect the upper and lower bounds calculated theoretically. The estimation of the average case is close to the simulations for a small number of groups. However, increasing the number of groups decreases the precision of the approximation, due to the minimization of the transition probabilities to get in the *Correct* state (see Section 7.4.2).

Figure 7.5(b) shows the cluster stability depending on the time history, for a network composed of 10 groups. We notice that increasing the time history considerably improves the cluster stability and the theoretical approximation. For a time history $h_{min} = 4$ ($H = 7$), a stability of 99.92 is achieved, while the lower bound is 99.89. Therefore, for achieving a stability close to 100%, the necessary delay is $H \times 16 = 112$ seconds ($H = 7$ and the size of the data sequence is 16 seconds, see Chapter 6).



(a) $h_{min} = H = 1$



(b) $h_{min} = 1..4$, 10 groups

Figure 7.5: Cluster stability in the RTI with tilt switches scenario.

7.5.2.2 RTI with accelerometers scenario.

The solution using accelerometers is more reliable than the method with tilt switches, resulting in higher probabilities for a correct detection of the context (see Table 7.2) and consequently, higher cluster stability. Figure 7.6 shows the cluster stability depending on the number of groups present in the network, given that the time history is $h_{min} = 1$. We also represent the error bars for the absolute minimum and maximum values. We notice the high stability obtained even for large number of groups (99.5 for 10 groups). Due to the fact that the clusters stay in the *Correct* state for most of the time, the approximations are close to the simulation results. For this scenario, a high cluster stability can be achieved even considering the time history 1, reaching a delay of only 16 seconds.

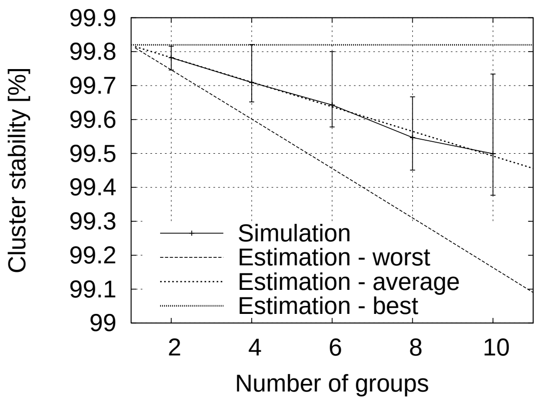
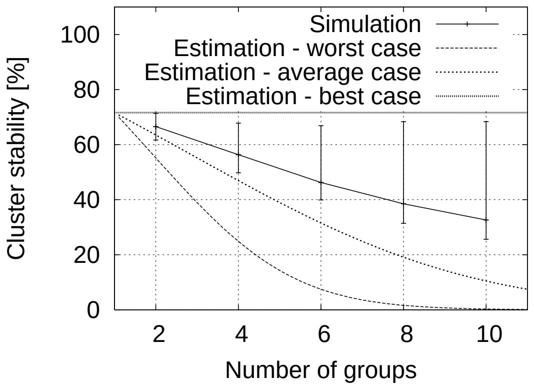


Figure 7.6: Cluster stability in the RTI with accelerometers scenario ($h_{min} = H = 1$).

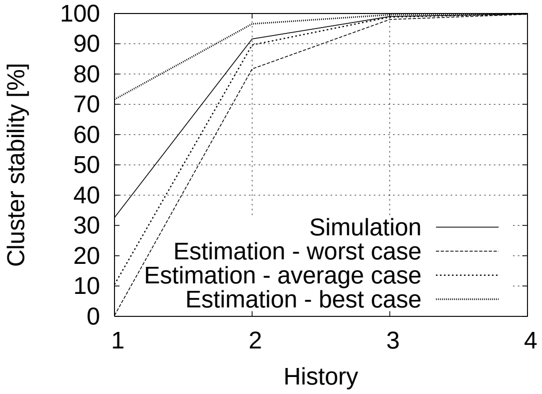
7.5.2.3 BAN scenario.

Figure 7.7(a) shows the cluster stability in the BAN scenario, depending on the number of groups, given that the time history is $h_{min} = 1$. Similarly with the two scenarios presented above, we notice that the results respect the upper and lower bounds calculated theoretically. The average stability is lower than in the previous cases, with a maximum of 67% and less than 50% for a network composed of more than 6 groups.

Figure 7.7(b) shows the cluster stability depending on the time history, for a network composed of 10 groups. The time history significantly improves the cluster stability: for the time history $h_{min} = 4$ ($H = 7$), the stability is 99.84 and the lower bound is 99.74. For achieving this, the delay is $H \times 8 = 56$ seconds ($H = 7$ and the window size is 8 seconds [109]).



(a) $h_{min} = H = 1$



(b) $h_{min} = 1..4$, 10 groups

Figure 7.7: Cluster stability in the BAN scenario.

7.5.2.4 Communication overhead.

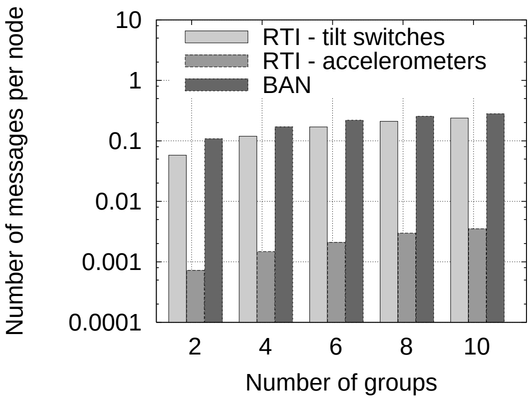
The communication overhead is induced by the *SetRoot* message, sent by every node when the current root node changes. In case the lower networking layers use a periodic message in order to maintain for example the synchronization (e.g. in case of a TDMA MAC protocol), Tandem can use this heartbeat to piggyback the small piece of information present in the *SetRoot* message (root address and priority number). Assuming that there is no such periodic message, we count the average number of *SetRoot* messages sent by each node in one step of simulation on average.

Figure 7.8(a) shows the number of messages on a logarithmic scale, depending on the number of groups. The results correspond to each of the three scenarios, where the time history is 1. We notice that the more stable the structure is, the less communication overhead is needed. For the RTI with accelerometers scenario, less than 1 message is sent per node in 100 time steps, even for a large number of groups. The overhead is increasing as the number of groups increases, due to the diminishing cluster stability.

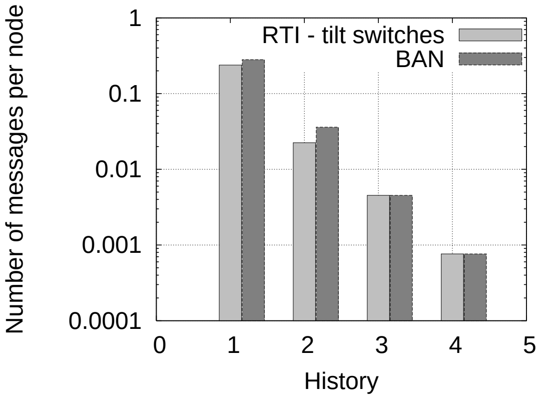
Figure 7.8(b) shows the number of messages depending on the time history, for the RTI with tilt switches and BAN scenarios. Increasing the time history improves the stability and thus reduces the communication overhead. For the time history 4, the overhead is less than 10^{-3} messages per node.

7.6 Discussion and conclusions

Tandem is a context-aware method for spontaneous clustering of wireless sensor nodes. The algorithm allows reclustering in case of topological or contextual changes. By analysing the similarity of the context over a time history, Tandem tries to achieve stable clusters. We approximate the behaviour of the algorithm using a Markov chain model and we analyse the cluster stability theoretically and through simulations, using experimental results from real field tests. The analysis gives the possibility to estimate theoretically the stability of the structure and the responsiveness of the algorithm. Computing the worse case stability via the Markov chain approximation, we can deduce the time history necessary to achieve stable clusters. In what follows, we discuss the main advantages and limitations of the proposed clustering method.



(a) Number of groups.



(b) History.

Figure 7.8: Average number of *SetRoot* messages sent per node in 100 steps.

7.6.1 Advantages

- *Responsiveness.* The clustering structure reacts quickly to topological and contextual changes: nodes decide based only on the current situation of their neighbourhood, without the need of any negotiation with other parties.

- *Learning system.* Since the probabilities p and q are assumed to be the same for each pair of nodes, only a small-scale reproducible experiment is required for computing these probabilities, which can be used to estimate the cluster stability. For example, two nodes moving together and another two moving separately are enough to generate the initial statistical distributions of the confidence values. The accuracy of the estimation can be further improved by performing experiments on a larger scale.

- *Delay estimation.* By deducing the time history required to achieve a certain stability, the delay in accommodating the topological or contextual changes can be easily estimated.

7.6.2 Limitations

- *Equality assumption.* Our solution for estimating the cluster stability depends on Assumption 8, which implies that probabilities p and q are the same for each pair of nodes. In case nodes do not experience the same movements (for example, in the RTI scenario, if they are placed on loose frames), these probabilities differ depending on the placement of nodes. The cluster stability can still be computed in this case, using the distribution of the confidence value in the worse case scenario (as we show in Section 7.5.2 for the BAN scenario). However, since we consider the worse case scenario, the stability estimated theoretically is lower than the real stability.

- *Rough approximation for many groups.* As we notice from Figures 7.5(a) and 7.7(a), the difference between the approximation that we derive using Markov chains and the real situation is increasing with the number of groups. However, the model offers a good approximation in case of highly accurate context detection methods (see Figure 7.6). Therefore, the approximation can be successfully used for deducing the minimum time history for a cluster stability close to 100%.

- *Multihop clusters.* The method that we propose is valid only for one-hop clusters, which is justified taking into account the scenarios from Section 7.2. Nevertheless, other applications may require multihop clusters, even several layers of clustering. For example, groups of people skiing together, forming multihop clusters, where each person is wearing a BAN that is a one-hop cluster. The algorithm can be extended to accommodate multihop clusters: instead of choosing directly the clusterhead node, every node selects a parent and thus joins the cluster associated with the parent node.

Tandem represents a means to achieve context-aware clusters that reproduce the physical reality, for example to actual configuration of the BAN (i.e. the sensor attached to the garments or handeled by the person). Subsequent to clustering, a task allocation mechanism is required to distribute various tasks to the nodes which are part of the cluster, depending on their capabilities. The final goal is to have a distributed activity recognition algorithm running on a dynamic, context-aware BAN.

Appendix

We present the numerical average values for the comparison between Tandem and DMAC* from Section 7.5.1, along with 5^{th} and 95^{th} percentiles.

Number of groups	**2**	**4**	**6**	**8**	**10**
Tandem - average value	90.07	87.05	83.54	80.99	77.6
Tandem - 5^{th} percentile	89.28	85.79	80.77	78.25	74.92
Tandem - 95^{th} percentile	90.86	88.97	88.64	85.36	80.78
DMAC* - average value	86.5	78.06	70.47	64.12	58.97
DMAC* - 5^{th} percentile	84.53	73.1	65.54	58.33	53.43
DMAC* - 95^{th} percentile	88.48	82.58	82.75	79.7	76.36

Table 7.3: Average cluster stability, with 5^{th} and 95^{th} percentiles (see Figure 7.4(a))

Number of groups	**2**	**4**	**6**	**8**	**10**
Tandem - average value	0.03	0.054	0.078	0.093	0.111
Tandem - 5^{th} percentile	0.018	0.035	0.026	0.059	0.086
Tandem - 95^{th} percentile	0.042	0.066	0.098	0.114	0.129
DMAC* - average value	0.038	0.075	0.106	0.137	0.164
DMAC* - 5^{th} percentile	0.023	0.049	0.039	0.028	0.029
DMAC* - 95^{th} percentile	0.051	0.101	0.13	0.169	0.195

Table 7.4: Average number of messages per node, with 5^{th} and 95^{th} percentiles (see Figure 7.4(b))

Chapter 8

Conclusions

From the environmental monitoring applications studied initially, WSN technology has advanced at a fast pace. Today's market offers mature platforms, including development tools and complete system support. New releases of sensor nodes show a clear trend of converging towards a uniformly accepted network standard, with IEEE 802.15.4 and ZigBee being the prominent options. The foreseen evolution of WSNs leads to interoperability, dynamics, further miniaturization, heterogeneity and pervasive usage, in other words to the realization of Marc Weiser's ubiquitous computing vision. Out of these characteristics, *dynamics* is an important system property, which needs to be considered from the protocol design phase.

This thesis focuses on a class of protocols and algorithms that are able to cluster efficiently in the presence of mobility, and even exploit it to increase the functionality of the network. The contributions of this thesis are the following:

1. **Classifications of service discovery protocols and clustering algorithms.** We review the state of the art service discovery protocols and clustering algorithms. In both cases, we follow three methodological steps: (1) define the problem, general objectives and properties, (2) classify the existing solutions with respect to the defined objectives and properties and (3) frame the state of the art in a comparative table according to the proposed classification.

2. **A generalized clustering algorithm for wireless sensor networks.** We propose a generalized clustering algorithm for dynamic sensor networks. Our generalization allows for a better understanding and com-

parison of algorithms designed in mobile WSN environments, and facilitates the definition and demonstration of common properties for such algorithms.

3. **Cluster-based service discovery for wireless sensor networks.** We propose a combined, cluster-based service discovery solution for heterogeneous and dynamic wireless sensor networks. The service discovery protocol exploits a cluster overlay for distributing the tasks according to the capabilities of the nodes and providing an energy-efficient search. The clustering algorithm is explicitly designed to function as a distributed service registry and represents a particular case of the general algorithm proposed by Contribution 2.

4. **On-line recognition of joint movement in wireless sensor networks.** We propose a method through which dynamic sensor nodes determine that they move together by communicating and correlating their movement information. The movement information is acquired from tilt switches and accelerometer sensors. We implement a fast, incremental correlation algorithm, which can run on resource constrained devices.

5. **A context-aware method for spontaneous clustering of dynamic wireless sensor nodes.** We propose a method through which wireless sensor nodes organize spontaneously into clusters based on a common context, such as movement information. This algorithm is a particular case of the general algorithm proposed in Contribution 2.

These contributions address the research question from Chapter 1 from two perspectives. Firstly, our lightweight service discovery protocol can assist a multitude of dynamic applications where clients can configure, discover and use a variety of services. Secondly, our two clustering algorithms are tailored to dynamic WSN. C4SD is integrated with the service discovery protocol for the purpose of minimizing the communication overhead and reducing the negative effect of mobility on the consistency of the service registries. Tandem is an algorithm that allows service provisioning in a mobile group-based environment, by spontaneously clustering nodes based on similar mobility patterns.

Future work may generalize the idea of spontaneous clustering based on a common context, by providing a multi-level hierarchy of self-organizing wireless sensor nodes. At each hierarchical level, nodes group themselves into clusters based on similar semantic properties. For example, at the lowest level, each cluster may reflect the physical arrangement of the associated BAN. Further on, the clusterhead nodes from this level may form a second hierarchical level, for instance, representing a group of people walking together or a working team. The more hierarchical levels, the larger the coverage of the structure. The major benefit of this clustering solution is that it mimics the structures present in the real world. Semantic, multi-level clustering allows context-aware, dynamic service provisioning and facilitates the aggregation of the functionality through successive levels of exploitation. To achieve this structure, we highlight two major research directions: (1) providing accurate shared-context recognition algorithms at each semantical level in the hierarchy, and (2) developing efficient resource management mechanisms, as the heterogeneity of resources and the dynamics of the environment require novel techniques of distributed processing and dynamic task allocation.

Author References

[1] R. S. Marin-Perianu, M. Marin-Perianu, P. J. M. Havinga, and J. Scholten. Movement-based group awareness with wireless sensor networks. In *Proceedings of the 5th International Conference on Pervasive Computing (Pervasive)*, pages 298–315. Springer Verlag, 2007.

[2] R. S. Marin-Perianu, J. L. Hurink, and P. H. Hartel. A generalized clustering algorithm for dynamic wireless sensor networks. In *Proceedings of the International Workshop on Modeling, Analysis and Simulation of Sensor Networks (MASSN)*. IEEE Computer Society, December 2008. To appear.

[3] R. S. Marin-Perianu, J. Scholten, P. J. M. Havinga, and P. H. Hartel. Energy-efficient cluster-based service discovery in wireless sensor networks. In *Proceedings of the 31st IEEE Conference on Local Computer Networks*, pages 931–938. IEEE Computer Society, November 2006.

[4] R. S. Marin-Perianu, J. Scholten, P. J. M. Havinga, and P. H. Hartel. Cluster-based service discovery for heterogeneous wireless sensor networks. *International Journal of Parallel, Emergent and Distributed Systems*, 23(4):325–346, August 2008.

[5] R. S. Marin-Perianu, J. Scholten, and P. J. M. Havinga. Prototyping service discovery and usage in wireless sensor networks. In *Proceedings of the 32nd IEEE Conference on Local Computer Networks*, pages 841–850. IEEE Computer Society, October 2007.

[6] R. S. Marin-Perianu, C. Lombriser, P. J. M. Havinga, J. Scholten, and G. Troster. Tandem: A context-aware method for spontaneous clustering of dynamic wireless sensor nodes. In *Proceedings of Internet of Things, the International Conference for Industry and Academia*, pages 342–360. Springer, 2008.

[7] R. S. Marin-Perianu, J. Scholten, and P. J. M. Havinga. CODE: Description language for wireless collaborating objects. In *Proceedings of the 2nd International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pages 169–174. IEEE Computer Society, December 2005.

[8] C. Lombriser, M. Marin-Perianu, R. S. Marin-Perianu, D. Roggen, P. J. M. Havinga, and G. Tröster. Organizing context information processing in dynamic wireless sensor networks. In *Proceedings of the 3rd International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pages 67–72. IEEE Computer Society, December 2007.

[9] C. Lombriser, R. S. Marin-Perianu, D. Roggen, P. J. M. Havinga, and G. Tröster. Modeling service-oriented context processing in dynamic body area networks. *IEEE Journal on Selected Areas in Communications, Special Issue on Body Area Networking: Technology and Applications.* To appear.

Demonstration Papers

[10] R. S. Marin-Perianu, J. Scholten, and P. J. M. Havinga. Demo abstract: Service oriented wireless sensor networks - a cluster-based approach. In *4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks, SECON '07, San Diego, California*, pages 698–699. IEEE Computer Society Press, June 2007.

[11] M. Marin-Perianu, R. S. Marin-Perianu, P. J. M. Havinga, and J. Scholten. Online movement correlation of wireless sensor nodes. In *Advances in Pervasive Computing - Adjunct Proceedings of Pervasive 2007, Toronto, Canada*, pages 79–82. Austrian Computer Society, May 2007.

Web References (Last accessed: June 2008)

[12] 29 Palms Fixed/Mobile Experiment: Tracking vehicles with a UAV-delivered sensor network. http://robotics.eecs.berkeley.edu/ pister/29Palms0103/.

[13] Ambient Systems. Enschede, The Netherlands. http://www.ambient-systems.net.

[14] ASSEMTECH CW1300-1 ball-contact tilt switch. http://www.farnell.com/datasheets/67723.pdf.

[15] Crossbow Wireless Sensor Networks, Product Reference Guide 2007. San Jose, CA, USA. http://www.xbow.com/.

[16] Darpa agent markup language (daml+oil). http://www.daml.org/.

[17] IEEE standard 802.15.4. http://standards.ieee.org.

[18] Internet assigned numbers authority (iana). http://www.iana.org/.

[19] Parani-ESD200 Bluetooth module. http://www.sena.com/download/datasheet/ds_parani_esd.pdf.

[20] Project Sun SPOT. Santa Clara, CA, USA. http://www.sunspotworld.com/.

[21] Resource description framework (rdf). http://www.w3.org/RDF/.

[22] Sensinode product catalog. Oulu, Finland. http://www.sensinode.com/.

[23] Sentilla. Redwood City, CA, USA. http://www.sentilla.com/.

[24] STMicroelectronics LIS3LV02DQ 3-axis linear accelerometer. http://www.st.com/stonline/products/literature/ds/11115.pdf.

[25] TECO Particle. Karlsruhe, Germany. http://particle.teco.edu/.

[26] XYZ Sensor Node. Yale University. http://www.eng.yale.edu/enalab/xyz/.

[27] ZigBee Alliance. http://www.zigbee.org.

References

[28] A. A. Abbasi and M. Younis. A survey on clustering algorithms for wireless sensor networks. *Comput. Commun.*, 30(14-15):2826–2841, 2007.

[29] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *Symposium on Operating Systems Principles*, pages 186–201, Charleston, SC, December 1999.

[30] I. Akyildiz and I. Kasimoglu. Wireless sensor and actor networks: research challenges. *Ad Hoc Networks*, 2(4):351–367, October 2004.

[31] I. F. Akyildiz, Su Weilian, Y. Sankarasubramaniam, and E. E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, 2002.

[32] A. D. Amis, R. Prakash, D. H., and T. Vuong. Max-min d-cluster formation in wireless ad hoc networks. In *INFOCOM*, pages 32–41. IEEE Computer Society Press, 2000.

[33] S. Antifakos, B. Schiele, and L. E. Holmquist. Grouping mechanisms for smart objects based on implicit interaction and context proximity. In *UBICOMP 2003 Interactive Posters*, pages 207 – 208, 2003.

[34] K. Arabshian and H. Schulzrinne. GloServ: Global service discovery architecture. In *MobiQuitous*, pages 319–325. IEEE Computer Society, June 2004.

[35] A. Avizienis, J.C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 01(1):11–33, 2004.

[36] R. Aylward and J. A. Paradiso. Sensemble: a wireless, compact, multi-user sensor system for interactive dance. In *Proceedings of the 6th Conference on New interfaces for Musical Expression (NIME '06)*, pages 134–139, 2006.

[37] R. Bader, M. Pinto, F. Spenrath, P. Wollmann, and F. Kargl. BigNurse: A Wireless Ad Hoc Network for Patient Monitoring. In *Pervasive Health Conference and Workshops*, pages 1–4. IEEE Computer Society, December 2006.

[38] A. Baggio. Wireless sensor networks in precision agriculture. In *Workshop on Real-World Wireless Sensor Networks (REALWSN 2005)*, Stockholm, Sweden, June 2005.

[39] M. Balazinska, H. Balakrishnan, and D. Karger. INS/Twine: A scalable peer-to-peer architecture for intentional resource discovery. In *International Conference on Pervasive Computing (Pervasive)*, pages 195–210, London, UK, August 2002. Springer-Verlag.

[40] H. Baldus, K. Klabunde, and G. Müsch. Reliable set-up of medical body-sensor networks. In *EWSN*, pages 353–363. Springer Verlag, 2004.

[41] S. Bandyopadhyay and E. Coyle. An energy efficient hierarchical clustering algorithm for wireless sensor networks. In *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1713– 1723. IEEE Computer Society, 2003.

[42] S. Basagni. Distributed and mobility-adaptive clustering for multimedia support in multi-hop wireless networks. In *Proceedings of Vehicular Technology Conference (VTC)*, pages 889–893. IEEE Computer Society, 1999.

[43] S. Basagni. Distributed clustering for ad hoc networks. In *Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN)*, pages 310–315, Washington, DC, USA, 1999. IEEE Computer Society.

[44] S. Basagni. Distributed clustering for ad hoc networks. In *Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN)*, pages 310–315, Washington, DC, USA, 1999. IEEE Computer Society.

[45] C. Bettstetter. *Mobility Modeling, Connectivity, and Adaptive Clustering in Ad Hoc Networks.* PhD thesis, Technische Universität München, Germany, October 2003.

[46] G. J. Bishop-Hurley, D. L. Swain, D. M. Anderson, P. Sikka, C. Crossman, and P. Corke. Virtual fencing applications: Implementing and testing an automated cattle control system. *Comput. Electron. Agric.*, 56(1):14–22, 2007.

[47] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[48] J. Blum, M. Ding, A. Thaeler, and X. Cheng. *Handbook of Combinatorial Optimization*, chapter Connected Dominating Set in Sensor Networks and MANETs, pages 329–369. Springer, 2005.

[49] F. Bouhafs, M. Merabti, and H. Mokhtar. A semantic clustering routing protocol for wireless sensor networks. In *Consumer Communications and Networking Conference*, pages 351– 355. IEEE Computer Society, 2006.

[50] N. Bulusu, J. Heidemann, and D. Estrin. Gps-less low cost outdoor localization for very small devices. *IEEE Personal Communications Magazine*, 7(5):28–34, October 2000.

[51] C. Cachin, J. Camenisch, M. Dacier, Y. Deswarte, J. Dobson, D. Horne, K. Kursawe, J.C. Laprie, J.C. Lebraud, D. Long, T. McCutcheon, J. Muller, F. Petzold, B. Pfitzmann, D. Powell, B. Randell, M. Schunter, V. Shoup, P. Verissimo, G. Trouessin, R.J. Stroud, M. Waidner, and I. Welch. Malicious and accidental fault tolerance in internet applications: Reference model and use cases. Technical Report LAAS report no. 00280, MAFTIA, Project IST-1999-11583, 2000.

[52] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communications and Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2(5):483–502, 2002.

[53] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. One ring to rule them all: Service discovery and binding in structured peer-to-peer overlay networks. In *Proceedings of the 10th SIGOPS European Workshop*, pages 140–145, Saint-Emilion, France, September 2002. ACM.

[54] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao. Habitat monitoring: application driver for wireless communications technology. *SIGCOMM Computer Communication Review*, 31(2 supplement):20–41, 2001.

[55] D. Chakraborty, A. Joshi, Y. Yesha, and T. Finin. Toward Distributed Service Discovery in Pervasive Computing Environments. *IEEE Transactions on Mobile Computing*, 5(2):97–112, February 2006.

[56] H. Chan and A. Perrig. Ace: An emergent algorithm for highly uniform cluster formation. In *Proceedings of the First European Workshop on Sensor Networks (EWSN)*, pages 154–171. Springer, January 2004.

[57] S. Chatterjea, T. Nieberg, N. Meratnia, and P. J. M. Havinga. A distributed and self-organizing scheduling algorithm for energy-efficient data aggregation in wireless sensor networks. Technical Report TR-CTIT-07-10, Enschede, February 2007.

[58] M. Chatterjee, S. K. Das, and D. Turgut. Wca: A weighted clustering algorithm for mobile ad hoc networks. *Cluster Computing*, 5(2):193–204, 2002.

[59] Y. P. Chen, A. L. Liestman, and J. Liu. *Ad Hoc and Sensor Networks, Wireless Networks and Mobile Computing, Volume 2*, volume 75, chapter Clustering Algorithms for Ad Hoc Wireless Networks, pages 145–164. Nova Science Publishers, 2005.

[60] Bluetooth Consortium. Specification of the bluetooth system core version 1.0 b: Part e, service discovery protocol (SDP), November 1999.

[61] The Salutation Consortium. Salutation architecture specification version 2.0c, June 1999. available online at http://www.salutation.org/.

[62] D. J. Cook and S. K. Das. How smart are our environments? an updated look at the state of the art. *Pervasive Mob. Comput.*, 3(2):53–73, 2007.

[63] S. Croce, F. Marcelloni, and M. Vecchio. Reducing power consumption in wireless sensor networks using a novel approach to data aggregation. *The Computer Journal*, 51(2):227–239, 2008.

[64] D.J. Daley and D. Vere-Jones. *An Introduction to the Theory of Point Processes.* Springer, 1988.

[65] F. C. Delicato, P. F. Pires, L. Pirmez, and L. F. Carmo. A service approach for architecting application independent wireless sensor networks. *Cluster Computing*, 8(2-3):211–221, 2005.

[66] B. Divecha, A. Abraham, C. Grosan, and S.Sanyal. Impact of Node Mobility on MANET Routing Protocols Models. *Journal of Digital Information Management*, 5(1):19–24, 2007.

[67] S. O. Dulman. *Data-centric architecture for wireless sensor networks.* PhD thesis, Enschede, The Netherlands, October 2005.

[68] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. In *Proceedings of the 5th symposium on Operating systems design and implementation (OSDI)*, pages 147–163, New York, NY, USA, 2002. ACM.

[69] A. Ephremides, J. Wieselthier, and D. Baker. A design concept for reliable mobile radio networks with frequency hopping signaling. *Proceedings of the IEEE*, 75(1):56–73, 1987.

[70] A. Erman-Tüysüz, L. F. W. van Hoesel, J. Wu, and P. J. M. Havinga. Enabling mobility in heterogeneous wireless sensor networks cooperating with uavs for mission-critical management. Technical Report TR-CTIT-08-14, Enschede, February 2008.

[71] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: scalable coordination in sensor networks. In *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 263–270, New York, NY, USA, 1999. ACM.

[72] G. Ferrari and O. K. Tonguz. Impact of Mobility on the BER Performance of Ad Hoc Wireless Networks. *IEEE Transactions on Vehicular Technology*, 56(1):271–286, 2007.

[73] UPnP Forum. UPnP device architecture version 1.0, June 2000. available online at http://www.upnp.org/.

[74] C. Frank and H. Karl. Consistency challenges of service discovery in mobile ad hoc networks. In *International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, pages 105–114, New York, USA, 2004. ACM Press.

[75] J. Gao and P. Steenkiste. Rendezvous points-based scalable content discovery with load balancing. In *Networked Group Communication*, pages 71–78. ACM Press, 2002.

[76] M. Gerla, T. J.Kwon, and G. Pei. On-demand routing in large ad hoc wireless networks with passive clustering. In *Wireless Communications and Networking Conference (WCNC)*, pages 100–105. IEEE Computer Society, 2000.

[77] L. Girod, V. Bychkovskiy, J. Elson, and D. Estrin. Locating tiny sensors in time and space: A case study. In *International Conference on Computer Design (ICCD)*, pages 214–219, Washington, DC, USA, 2002. IEEE Computer Society.

[78] A. Gluhak, M. Presser, L. Zhu, S. Esfandiyari, and S. Kupschick. Towards mood based mobile services and applications. In *2nd European Conference on Smart Sensing and Context (EuroSSC)*, pages 159–174. Springer Verlag, 2007.

[79] C. M. Grinstead and J. L. Snell. *Introduction to Probability: Second Revised Edition*, chapter Markov Chains, pages 405–470. American Mathematical Society, 1997.

[80] L. Gu, D. Jia, P. Vicaire, T. Yan, L. Luo, A. Tirumala, Q. Cao, T. He, J. A. Stankovic, T. Abdelzaher, and B. H. Krogh. Lightweight detection and classification for wireless sensor networks in realistic environments. In *SenSys*, pages 205–217, New York, NY, USA, 2005. ACM Press.

[81] E. Guttman and C. Perkins. Service location protocol, version, June 1999.

[82] B. Han and W. Jia. Clustering wireless ad hoc networks with weakly connected dominating set. *Journal of Parallel and Distributed Computing*, 67(6):727–737, 2007.

[83] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *Wireless Communications, IEEE Transactions on*, 1(4):660–670, 2002.

[84] S. Helal, N. Desai, V. Verma, and C. Lee. Konark – A Service Discovery and Delivery Protocol for Ad-Hoc Networks. In *Proceedings of the Third IEEE Conference on Wireless Communication Networks (WCNC)*, pages 2107–2113. IEEE Computer Society, March 2003.

[85] A. Helmy. *Resource Management in Wireless Networking*, volume 16, chapter Efficient Resource Discovery in Wireless AdHoc Networks: Contacts Do Help, pages 419–471. Springer, 2005.

[86] T. D. Hodes, S. E. Czerwinski, B. Y. Zhao, A. D. Joseph, and R. H. Katz. An architecture for secure wide-area service discovery. *Wireless Networks*, 8(2/3):213–230, 2002.

[87] L. Van Hoesel. *Schedule-Based Medium Access Control Protocols for Wireless Sensor Networks*. PhD thesis, University of Twente, 2007.

[88] 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.

[89] L. E. Holmquist, F. Mattern, B. Schiele, P. Alahuhta, M. Beigl, and H. Gellersen. Smart-its friends: A technique for users to easily establish connections between smart artefacts. In *UbiComp'01*, pages 116–122, London, UK, 2001. Springer-Verlag.

[90] T. A. Howes, M. C. Smith, and G. S. Good. *Understanding and Deploying LDAP Directory Services*. Macmillan Network Architecture and Development Series. Macmillan Technical Publishing, second edition, 2003.

[91] Y. J., L. W., Y. Kim, and X. Yang. Eemc: An energy-efficient multi-level clustering algorithm for large-scale wireless sensor networks. *Computer Networks*, 52(3):542–562, 2008.

[92] C. Johnen and L. H. Nguyen. Self-stabilizing weight-based clustering algorithm for ad hoc sensor networks. In *Proceedings of the Second International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS)*, pages 83–94. Springer, 2006.

[93] D. Johnson, D. Maltz, and J. Broch. *Ad Hoc Networking*, chapter 5, pages 139–172. Addison-Wesley, 2001.

[94] D. B Johnson and D. A Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353, pages 153–181. Kluwer Academic Publishers, 1996.

[95] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebranet. *SIGPLAN Not.*, 37(10):96–107, 2002.

[96] M. Kam, Z. Xiaoxun, and P. Kalata. Sensor fusion for mobile robot navigation. *Proceedings of the IEEE*, 85(1):108–119, January 1997.

[97] J. Kang, D. Kim, and S. Ahn. Mosaic localization for wireless sensor networks. In *Wireless Communications and Networking Conference(WCNC)*, pages 3924–3928. IEEE Computer Society, 2007.

[98] R. Kauw-A-Tjoe, J. Thalen, M. Marin-Perianu, and P. J. M. Havinga. Sensorshoe: Mobile gait analysis for parkinson's disease patients. In *UbiComp 2007 Workshop Proceedings, Innsbruck, Austria*, pages 187–191. University of Innsbruck, September 2007.

[99] M. Klein, B. König-Ries, and P. Obreiter. Lanes - a lightweight overlay for service discovery in mobile ad hoc networks. Technical Report 2003-6, University of Karlsruhe, 2003.

[100] M. Klein, B. König-Ries, and P. Obreiter. Service rings - a semantic overlay for service discovery in ad hoc networks. In *Proceedings of the 14th International Workshop on Database and Expert Systems Applications (DEXA)*, pages 180–185, Washington, DC, USA, 2003. IEEE Computer Society.

[101] R.M. Kling. Intel motes: advanced sensor network platforms and applications. *Microwave Symposium Digest, 2005 IEEE MTT-S International*, page 4, 12-17 June 2005.

[102] U. C. Kozat and L. Tassiulas. Service discovery in mobile ad hoc networks: An overall perspective on architectural choices and network layer support issues. *Ad Hoc Networks*, 2(1):23–44, June 2003.

[103] M. Kumar and S. K. Das. *Handbook of Nature-Inspired and Innovative Computing*, chapter Pervasive Computing: Enabling Technologies and Challenges, pages 613–631. Springer, 2006.

[104] K. Van Laerhoven and H. Gellersen. Spine versus porcupine: A study in distributed wearable activity recognition. In *Proceedings of the Eighth International Symposium on Wearable Computers (ISWC'04)*, pages 142–149, 2004.

[105] G. H. K. Lam, H. V. Leong, and S. C. Chan. Gbl: Group-based location updating in mobile environment. In *9th International Conference on Database Systems for Advanced Applications (DASFAA)*, pages 762–774, 2004.

[106] Y. W. Law. *Key management and link-layer security of wireless sensor networks : Energy-efficient attack and defense.* PhD thesis, University of Twente, December 2005.

[107] C. Lee and S. Helal. A multi-tier ubiquitous service discovery protocol for mobile clients. In *Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, pages 701–711, Montréal, Canada, 2003.

[108] V. Lenders, M. May, and B. Plattner. Service discovery in mobile ad hoc networks: A field theoretic approach. *Pervasive and Mobile Computing*, 1:343–370, 2005.

[109] J. Lester, B. Hannaford, and G. Borriello. "Are You with Me?" - using accelerometers to determine if two devices are carried by the same person. In *Pervasive*, pages 33–50. Springer Verlag, 2004.

[110] A. Leung and C. Mitchell. A service discovery threat model for ad hoc networks. In Manu Malek, Eduardo Fernndez-Medina, and Javier Hernando, editors, *SECRYPT*, pages 167–174. INSTICC Press, 2006.

[111] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. *Ambient Intelligence*, chapter TinyOS: An Operating System for Sensor Networks, pages 115–148. Springer Berlin Heidelberg, 2005.

[112] W. Li. Random texts exhibit Zipf's law-like word frequency distribution. *IEEETIT: IEEE Transactions on Information Theory*, 38(6), 1992.

[113] M. E. M. Lijding, N. Meratnia, H. P. Benz, and A. Matysiak Szóstek. Smart signs show you the way. *I/O Vivat*, 22(4):35–38, August 2007.

[114] C. R. Lin and M. Gerla. Adaptive clustering for mobile wireless networks. *Selected Areas in Communications, IEEE Journal on*, 15(7):1265–1275, 1997.

[115] K. Lorincz, D. J. Malan, T. R.F. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, M. Welsh, and S. Moulton. Sensor networks for emergency response: Challenges and opportunities. *IEEE Pervasive Computing*, 03(4):16–23, 2004.

[116] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 491–502, New York, NY, USA, 2003. ACM.

[117] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97, New York, NY, USA, 2002. ACM.

[118] 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.

[119] M. Marin-Perianu, C. Lombriser, O. Amft, P. J. M. Havinga, and G. Troster. Distributed activity recognition with fuzzy-enabled wireless sensor networks. In *Distributed Computing in Sensor Systems, Santorini, Greece*, pages 296–313. Springer Verlag, June 2008.

[120] M. Marin-Perianu, N. Meratnia, P. J. M. 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.

[121] M. Marin-Perianu, N. Meratnia, M. Lijding, and P. Havinga. Being aware in wireless sensor networks. In *15th IST Mobile and Wireless Communication Summit, Capturing Context and Context Aware Systems and Platforms Workshop*, 2006.

[122] M. Marin-Perianu and P. J. M.Havinga. D-FLER: A distributed fuzzy logic engine for rule-based wireless sensor networks. In *4th International Symposium on Ubiquitous Computing Systems (UCS)*, pages 86–101. Springer Verlag, 2007.

[123] R. Mayrhofer and H. Gellersen. Shake well before use: Authentication based on accelerometer data. In *Pervasive*, pages 144–161. Springer-Verlag, 2007.

[124] A. McDonald and T. Znati. A mobility-based framework for adaptive clustering in wireless ad hoc networks. *IEEE Journal on Selected Areas in Communications (JSAC)*, 17(8):1466–1486, August 1999.

[125] V. P. Mhatre, C. Rosenberg, D. Kofman, R. Mazumdar, and N. Shroff. A minimum cost heterogeneous sensor network with a lifetime constraint. *IEEE Transactions on Mobile Computing*, 4(1):4–15, 2005.

[126] F. Michahelles, P. Matter, A. Schmidt, and B. Schiele. Applying wearable sensors to avalanche rescue. *Computers and Graphics*, 27(6):839–847, 2003.

[127] Sun Microsystems. Jini architecture specification version 2.0, June 2003.

[128] Sun Microsystems. Jini technology surrogate architecture specification, October 2003.

[129] P. Mockapetris and K. J. Dunlap. Development of the domain name system. *SIGCOMM Comput. Commun. Rev.*, 18(4):123–133, 1988.

[130] A. Molisch. *Wireless Communications*. John Wiley and Sons, 2005.

[131] M. Nidd. Service discovery in DEAPspace. *ieee-pcm*, 8(4):39–45, August 2001.

[132] T. Nieberg. *Independent and Dominating Sets in Wireless Communication Graphs*. PhD thesis, University of Twente, April 2006.

[133] F. G. Nocetti, J. S. Gonzalez, and I. Stojmenovic. Connectivity based k-hop clustering in wireless networks. *Telecommunication Systems*, 22(1–4):205–220, 2003.

[134] J. P., J. Hill, and D. Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems (SenSys)*, pages 95–107, New York, NY, USA, 2004. ACM.

[135] G. Pang and H. Liu. Evaluation of a low-cost mems accelerometer for distance measurement. *Journal of Intelligent and Robotic Systems*, 30(3):249–265, 2001.

[136] C. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *SIGCOMM*, pages 234–244. ACM press, 1994.

[137] C. Perkins and E. Royer. Ad-hoc on-demand distance vector routing. In *Workshop on Mobile Computing Systems and Applications (WMCSA)*, pages 90–100, Los Alamitos, CA, USA, 1999. IEEE Computer Society.

[138] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Schenker. A scalable content-addressable network. Technical Report TR-00-010, Berkeley, CA, August 2001.

[139] O. V. Ratsimor, D. Chakraborty, S. Tolia, D. Khushraj, A. Kunjithapatham, A. Joshi, T. Finin, and Y. Yesha. Allia: Alliance-based service discovery for ad-hoc environments. In *ACM Mobile Commerce Workshop*, pages 1–9, September 2002.

[140] R. Robinson and J. Indulska. Superstring: A scalable service discovery protocol for the wide area pervasive environment. In *Proc. Of the 11th IEEE International Conference on Networks, Sydney*, pages 699–704, Sydney, September 2003. Proc. of the 11th IEEE International Conference on Networks.

[141] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, November 2001.

[142] K. Sha, W. Shi, and O. Watkins. Using wireless sensor networks for fire rescue applications: Requirements and challenges. In *IEEE International Conference on Electro/Information Technology*, pages 239–244. IEEE Computer Society, May 2006.

[143] R. Shirey. Request for comments: 2828, May 2000. Internet Security Glossary.

[144] K. Sohrabi, J. Gao, V. Ailawadhi, and G.J. Pottie. Protocols for self-organization of a wireless sensor network. *IEEE Personal Communications*, 7(5):16–27, Oct 2000.

[145] A. Somasundara, A. Kansal, D. Jea, D. Estrin, and M. Srivastava. Controllably mobile infrastructure for low energy embedded networks. *IEEE Transactions on Mobile Computing*, 5(8):16, 2006.

[146] T. Stiefmeier, C. Lombriser, D. Roggen, H. Junker, G. Ogris, and G. Tröster. Event-based activity tracking in work environments. In *Proceedings of the 3rd International Forum on Applied Wearable Computing (IFAWC)*, March 2006.

[147] I. Stoica, R. Morris, D. R. Karger, M. Frans Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, August 2001.

[148] I. Stojmenovic, M. Seddigh, and J. Zunic. Dominating sets and neighbor elimination-based broadcasting algorithms in wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):14–25, 2002.

[149] D. Stoyan, W. S. Kendall, and J. Mecke. *Stochastic Geometry and its Applications*. John Wiley and Sons, 1995.

[150] M. Strohbach and H. Gellersen. Smart clustering - networking smart objects based on their physical relationships. In *Proceedings of the 5th IEEE International Workshop on Networked Appliances*, pages 151– 155. IEEE Computer Society, 2002.

[151] V. Sundramoorthy. *At Home in Service Discovery.* PhD thesis, Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, Enschede, Netherlands, September 2006.

[152] V. Sundramoorthy, J. Scholten, P. G. Jansen, and P. H. Hartel. Service discovery at home. In *4th Int. Conf. On Information, Communications & Signal Processing and 4th IEEE Pacific-Rim Conf. On Multimedia (ICICS/PCM)*, pages 1929–1933. IEEE Computer Society, December 2003.

[153] C.W. Tan, S. Park, K. Mostov, and P. Varaiya. Design of gyroscope-free navigation systems. In *Intelligent Transportation Systems, Oakland, USA*, pages 286–291, 2001.

[154] A. S. Tanenbaum, C. Gamage, and B. Crispo. Taking sensor networks from the lab to the jungle. *IEEE Computer*, 39(8):98–100, Aug 2006.

[155] B. Traversat, M. Abdelaziz, and E. Pouyoul. Project JXTA: A loosely-consistent DHT rendezvous walker, May 2003. Sun Microsystems, Inc.

[156] W. T. Tsai, C. V. Ramamoorthy, W. K. Tsai, and O. Nishiguchi. An adaptive hierarchical routing protocol. *IEEE Transactions on Computers*, 38(8):1059–1075, 1989.

[157] A. Varga. The OMNeT++ discrete event simulation system. In *European Simulation Multiconference (ESM)*, Prague, Czech Republic, June 2001.

[158] A. Varshavsky, B. Reid, and E. de Lara. A cross-layer approach to service discovery and selection in manets. In *Mobile Adhoc and Sensor Systems Conference (MASS)*, pages 8 pp.–. IEEE Computer Society, November 2005.

[159] P. Wan, K. M. Alzoubi, and O. Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. *Mobile Networks and Applications*, 9(2):141–149, 2004.

[160] C. Wang, K. Sohraby, B. Li, M. Daneshmand, and Y. Hu. A survey of transport protocols for wireless sensor networks. *IEEE Network*, 20(3):34–40, 2006.

[161] J. Wang, G. Chen, and D. Kotz. A sensor fusion approach for meeting detection. In *MobiSys 2004 Workshop on Context Awareness*, June 2004.

[162] K. Wang, S. A. Ayyash, T. Little, and P. Basu. Attribute-based clustering for information dissemination in wireless sensor networks. In *Proceedings of the Second Conference on Sensor and Ad Hoc Communications and Networks (SECON)*, pages 498–509. IEEE Computer Society, 2005.

[163] B. Warneke, M. Last, B. Liebowitz, and K. S. J. Pister. Smart dust: communicating with a cubic-millimeter computer. *Computer*, 34(1):44–51, 2001.

[164] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, September 1991.

[165] J. Wu and H. Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *Proceedings of the 3rd international workshop on Discrete algorithms and methods for mobile computing and communications (DIALM)*, pages 7–14, New York, NY, USA, 1999. ACM.

[166] K. Xu, Y. Wang, and Y. Liu. A clustering algorithm based on power for wsns. In *Proceedings of the International Conference on Computational Science (ICCS)*, pages 153–156. Springer, 2007.

[167] O. Younis and S. Fahmy. Heed: A hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks. *IEEE Transactions on Mobile Computing*, 3(4):366–379, 2004.

[168] A. M. Youssef, M. F. Younis, M. Youssef, and A. K. Agrawala. Distributed formation of overlapping multi-hop clusters in wireless sensor networks. In *Proceedings of the Global Telecommunications Conference (GLOBECOM)*, pages 1–6, 2006.

[169] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.

[170] F. Zhu, M. W. Mutka, and L. M. Ni. Splendor: A secure, private, and location-aware service discovery protocol supporting mobile services. In *PerCom*, pages 235–242, New York, NY, USA, 2003. ACM Press.

Titles in the IPA Dissertation Series since 2002

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects.* Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems.* Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems.* Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity.* Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems.* Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

N. van Vugt. *Models of Molecular Computing.* Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems.* Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

R. van Stee. *On-line Scheduling and Bin Packing.* Faculty of Mathematics and Natural Sciences, UL. 2002-09

D. Tauritz. *Adaptive Information Filtering: Concepts and Algorithms.* Faculty of Mathematics and Natural Sciences, UL. 2002-10

M.B. van der Zwaag. *Models and Logics for Process Algebra.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

J.I. den Hartog. *Probabilistic Extensions of Semantical Models.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12

L. Moonen. *Exploring Software Systems.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

J.I. van Hemert. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining.* Faculty of Mathematics and Natural Sciences, UL. 2002-14

S. Andova. *Probabilistic Process Algebra.* Faculty of Mathematics and Computer Science, TU/e. 2002-15

Y.S. Usenko. *Linearization in μCRL.* Faculty of Mathematics and Computer Science, TU/e. 2002-16

J.J.D. Aerts. *Random Redundant Storage for Video on Demand.* Faculty of Mathematics and Computer Science, TU/e. 2003-01

M. de Jonge. *To Reuse or To Be Reused: Techniques for component* *composition and construction.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-02

J.M.W. Visser. *Generic Traversal over Typed Source Code Representations.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-03

S.M. Bohte. *Spiking Neural Networks.* Faculty of Mathematics and Natural Sciences, UL. 2003-04

T.A.C. Willemse. *Semantics and Verification in Process Algebras with Data and Timing.* Faculty of Mathematics and Computer Science, TU/e. 2003-05

S.V. Nedea. *Analysis and Simulations of Catalytic Reactions.* Faculty of Mathematics and Computer Science, TU/e. 2003-06

M.E.M. Lijding. *Real-time Scheduling of Tertiary Storage.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-07

H.P. Benz. *Casual Multimedia Process Annotation – CoMPAs.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-08

D. Distefano. *On Modelchecking the Dynamics of Object-based Software: a Foundational Approach.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-09

M.H. ter Beek. *Team Automata – A Formal Approach to the Modeling of Collaboration Between System Components.* Faculty of Mathematics and Natural Sciences, UL. 2003-10

D.J.P. Leijen. *The λ Abroad – A Functional Approach to Software Components.* Faculty of Mathematics and Computer Science, UU. 2003-11

W.P.A.J. Michiels. *Performance Ratios for the Differencing Method.* Faculty of Mathematics and Computer Science, TU/e. 2004-01

G.I. Jojgov. *Incomplete Proofs and Terms and Their Use in Interactive Theorem Proving.* Faculty of Mathematics and Computer Science, TU/e. 2004-02

P. Frisco. *Theory of Molecular Computing – Splicing and Membrane systems.* Faculty of Mathematics and Natural Sciences, UL. 2004-03

S. Maneth. *Models of Tree Translation.* Faculty of Mathematics and Natural Sciences, UL. 2004-04

Y. Qian. *Data Synchronization and Browsing for Home Environments.* Faculty of Mathematics and Computer Science and Faculty of Industrial Design, TU/e. 2004-05

F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-06

L. Cruz-Filipe. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications.* Faculty of Science, Mathematics and Computer Science, KUN. 2004-07

E.H. Gerding. *Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals, Strategies, and Business Applications.* Faculty of

Technology Management, TU/e. 2004-08

N. Goga. *Control and Selection Techniques for the Automated Testing of Reactive Systems.* Faculty of Mathematics and Computer Science, TU/e. 2004-09

M. Niqui. *Formalising Exact Arithmetic: Representations, Algorithms and Proofs.* Faculty of Science, Mathematics and Computer Science, RU. 2004-10

A. Löh. *Exploring Generic Haskell.* Faculty of Mathematics and Computer Science, UU. 2004-11

I.C.M. Flinsenberg. *Route Planning Algorithms for Car Navigation.* Faculty of Mathematics and Computer Science, TU/e. 2004-12

R.J. Bril. *Real-time Scheduling for Media Processing Using Conditionally Guaranteed Budgets.* Faculty of Mathematics and Computer Science, TU/e. 2004-13

J. Pang. *Formal Verification of Distributed Systems.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-14

F. Alkemade. *Evolutionary Agent-Based Economics.* Faculty of Technology Management, TU/e. 2004-15

E.O. Dijk. *Indoor Ultrasonic Position Estimation Using a Single Base Station.* Faculty of Mathematics and Computer Science, TU/e. 2004-16

S.M. Orzan. *On Distributed Verification and Verified Distribution.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-17

M.M. Schrage. *Proxima - A Presentation-oriented Editor for Structured Documents.* Faculty of Mathematics and Computer Science, UU. 2004-18

E. Eskenazi and A. Fyukov. *Quantitative Prediction of Quality Attributes for Component-Based Software Architectures.* Faculty of Mathematics and Computer Science, TU/e. 2004-19

P.J.L. Cuijpers. *Hybrid Process Algebra.* Faculty of Mathematics and Computer Science, TU/e. 2004-20

N.J.M. van den Nieuwelaar. *Supervisory Machine Control by Predictive-Reactive Scheduling.* Faculty of Mechanical Engineering, TU/e. 2004-21

E. Ábrahám. *An Assertional Proof System for Multithreaded Java -Theory and Tool Support- .* Faculty of Mathematics and Natural Sciences, UL. 2005-01

R. Ruimerman. *Modeling and Remodeling in Bone Tissue.* Faculty of Biomedical Engineering, TU/e. 2005-02

C.N. Chong. *Experiments in Rights Control - Expression and Enforcement.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03

H. Gao. *Design and Verification of Lock-free Parallel Algorithms.* Faculty of Mathematics and Computing Sciences, RUG. 2005-04

H.M.A. van Beek. *Specification and Analysis of Internet Applications.* Faculty of Mathematics and Computer Science, TU/e. 2005-05

M.T. Ionita. *Scenario-Based System Architecting - A Systematic Approach to*