A Self-organizing Management Platform for Wireless Sensor Networks

Trang Cao Minh

PhD Thesis UPF / July 2014

SUPERVISOR
Dr. Boris Bellalta and Dr. Miquel Oliver

Department of Information and Communication Technologies

To my mother

Acknowledgments

I would like to acknowledge many people who helped me during the PhD course. First, I would like to thank my PhD supervisor, Prof. Boris Bellalta for giving a lot of good advice in my research. Prof. Boris believed and encouraged me to follow the research ideas with sensor network management systems. I would also like to thank my co-supervisor Prof. Miquel Oliver for his support and advice through this research.

I extend my gratitude to my colleagues Ruizhi Liao and Simon Oechsner for providing technical comments and feedback during my study. I am thankful to other members of my research group for a pleasant working environment. I would also like to thank to all of my friends for helping me a lot during my stay in Barcelona. I would like to especially mention some of the wonderful friends Huan, Cuong, Thuan, Thuy, Chinh, Vanessa, Jana, Magda, Jonathan.

My PhD studies were partly funded by the Spanish Government under project TEC2012-32354, by the Catalan Government (SGR-2014-1173), and the UPF PhD program.

Finally, I would like to say a great thank to my family, especially my mother. Without their encouragement and support, these studies would not have been possible.

Abstract

Through advances in sensor, networking, semiconductor and energy storage technologies, Wireless Sensor Networks (WSNs) are increasingly being deployed in many important applications to enable users to access information about the physical world. As a result, WSNs role as one of key components of the Internet Of Things. However, the increase of number of sensors, the number of sensor types, and the number of applications will make very difficult for the management systems of WSNs.

In this thesis, we propose a self organizing management platform designed to ensure sensor nodes and user applications are set up and running as intended. Our management platform, called DISON (DIstributed Self Organizing Network) uses a multilevel management schema to provide scalability for large sensor networks. We show how sensor nodes self adapt to the changes in network resources and application requirements and how network resources are coordinated efficiently among groups of adjacent sensor nodes.

For flexibility, our platform are implemented and performed in an independent layer and interact with the user application and network protocols through public interfaces. This helps our platform to be easily integrated to an existing or a new application. A set of management data models and protocols are developed to validate the efficiency of the proposed platform in resolving challenging management problems in both single and shared sensor networks.

Finally, in order to qualitatively evaluate our platform, we present two case studies, one with a single sensor network and another with a shared sensor network, where DISON was used to coordinate network resources and application requirements. The results from extensive experiments show that using DISON can bring a dramatic improvement to the scalability, the stability, the efficiency, the reliability and the flexibility of WSNs.

Resum

Gracias a los avances en la tecnología de sensores, redes, semiconductores y almacenamiento de energía, el uso de las redes de sensores inalámbricos (WSNs) ha augmentado notablemente en los últimos años con la aparación de multitud de nuevas aplicaciones que permiten a los usuarios acceder a la información sobre el mundo físico. Como resultado, las WSNs son uno de los componentes clave en la Internet of Things. Sin embargo, el aumento del número de sensores, la diversidad de sensores, y del número de aplicaciones hará muy difícil para los sistemas de gestión de redes inalámbricas de sensores.

En esta tesis, se propone una plataforma de gestión auto-organizativa diseñada para asegurar que los nodos sensores y aplicaciones de usuario estén siempre configurados y funcionando según lo previsto. Nuestra plataforma de gestión, llamada DISON (DIstributed Self Organizing Network), utiliza un esquema de gestión de varios niveles para proporcionar escalabilidad en redes de sensores de gran tamaño. Mostramos cómo los nodos sensores se auto-adaptan a los cambios en los recursos de la red y en los requisitos de las aplicaciones y cómo los recursos de red se coordinan de manera eficiente entre grupos de nodos de sensores adyacentes.

Para una mayor flexibilidad, nuestra plataforma se implementa y materializa en una capa independiente que interactúa con la aplicación de usuario y los protocolos de red a través de interfaces públicas. Esto ayuda a nuestra plataforma a integrarse fácilmente a una aplicación ya existente o a una nueva. Un conjunto de modelos de datos y protocolos de gestión se han desarrollado para validar la eficacia de la plataforma propuesta en la resolución de problemas de gestión de desafío en tanto las redes individuales y compartidas de los sensores.

Por último, con el objetivo de evaluar nuestra plataforma, presentamos dos estudios de caso, uno con una red individual de sensores y otra con una red compartidas de sensores, donde se utilizó DISON para coordinar los recursos de red y aplicaciones de usuario. Los resultados de extensos experimentos muestran que el uso de DISON puede aportar una mejora

dramática a la escalabilidad, la estabilidad, la eficiencia, la fiabilidad y la flexibilidad de WSN.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

DISON Distributed Self Organizing Network

GL Group Leave

HOS Smart Home, Smart Building - Home and Office automation Systems

ITS Intelligent Transportation Systems

MEMS Micro Electro Mechanical Systems

PDR Packet Delivery Rate

QoS Quality of Service

ROC Role Confirm

ROD Role Deny

ROF Role Offer

SG Smart Grid

TREG Task Register

TREP Task Reply

WSNs Wireless Sensor Networks

Chapter 1

INTRODUCTION

This chapter describes the demand for efficient self organizing management architectures for Wireless Sensor Networks (WSNs). This chapter is organized as follows: section 1.1 discusses background information; section 1.2 motivates the work proposed in this thesis; section 1.3 outlines the main research objectives; section 1.4 presents the approach adopted in the research; section 1.5 presents a summary of main results and contributions of this thesis, and finally section 1.6 presents the structure of the thesis.

1.1 Background

Cities are getting increasingly crowded. Many research groups, in both academia and industry, have recently put huge efforts to make cities smarter to ensure public safety, provide efficient transport, save energy, reduce expenses, and improve the quality of life. With advances in wireless communications and MEMS (Micro Electro Mechanical Systems), Smart Cities [1] [1] [2] [3] are becoming a reality. Three most commonly deployed

[1] In [1], a city can be defined as ’smart’ when investments in human and social capital and traditional (transport) and modern (ICT) communication infrastructure fuel sustainable economic development and a high quality of life, with a wise management of natural resources, through participatory action and engagement.

Smart Cites' applications are shown in Figure 1.1 and can be summarized as follows:

- **Intelligent Transportation Systems (ITS).** Intelligent transportation systems are applications which apply advances in information and communication technologies with the goal to organize traffic more efficiently, enhance safety and reduce CO2 emissions in transport systems. They can be deployed in vehicles (e.g., car, train, ship, and air plane) and infrastructure (e.g., road, train station, and gas station).

- **Smart Grids (SG).** The growing population has created a greater demand for energy while the limited amount of fossil fuels is diminishing. Additionally, the power grids designed and deployed in the past are not able to cope with current and future needs. To resolve these problems, smarter electrical grids which use information and communication technologies to optimize the energy distribution and to improve the efficiency and productivity of the energy usage are being developed. New smart grids can also help suppliers and consumers to monitor and control the energy usage and costs.

- **Smart Home, Smart Building - Home and Office automation Systems (HOS).** Home and office automation systems interconnect electric devices such as heaters, lights, air conditioners, TVs, computers, alarms, and cameras through a communication network, allowing them to be remotely controlled, monitored or accessed from any room in the building, as well as from any location in the world by Internet. They help people to optimize their living style, arrange the day-to-day schedule, secure a high living quality, and reduce the energy consumption bills.

Recently, there are numerous research projects aiming at the development of technologies for such cities, such as Open Cities [4] and Smart Santander [5]. In the Open Cities project [4], several European cities such as Amsterdam, Barcelona, Berlin, Helsinki, Paris are working on exploring Open and User Driven Innovation methodologies to the Public Sector

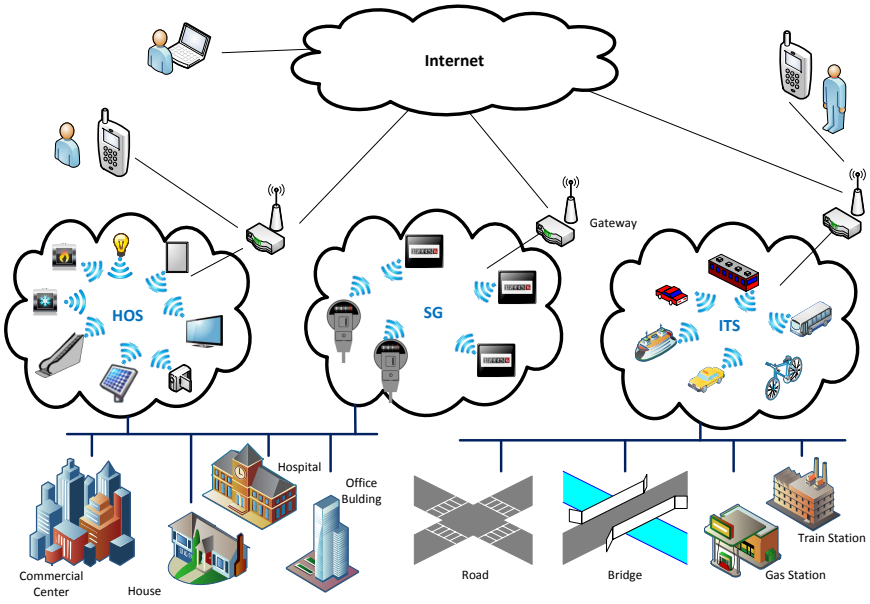


Figure 1.1: Smart City

in a scenario of Future Internet Services for Smart Cities. The Smart Santander project [5] focuses on designing, deploying and validating an experimental research facility to support typical applications and services for a smart city. In most of Smart Cities projects, wireless sensor networks play an important role in building instrumented and interconnected urban environments.

Wireless Sensor Networks (WSNs) are made up of small, low power, and low cost automated devices (i.e., sensor nodes), which have the capability of sensing, data processing, and wireless communication. Given these capabilities, WSNs can be deployed in different environments and using a large number of sensor nodes with an affordable cost. WSNs have wide applications in a variety of areas from industry, military to medical, scientific. Examples of applications include habitat monitoring, structure monitoring, smart homes and offices, surveillance, intelligent transporta-

tion systems, and many others [6] [7] [8] [9][10]. Therefore, WSNs are one of the critical components of Smart Cities. For example, in HOS, wireless sensor nodes are used to monitor or detect temperature, light, gas leaks and fire. The output of these sensors can be used to adjust the operation of electric appliances at homes. In ITS, numerous sensors installed on a vehicle can detect obstacles, measure the speed of the leading vehicle, warn impending collisions to the driver and trigger the collision avoidance system when necessary. Infrastructure sensors including induction loops, video and image processing, and microwave radars can be installed on the road to monitor traffic conditions and detect traffic congestion. Several recent studies have examined the feasibility of using WSNs in ITS. Wang et al. in [9], designed and implemented EasiTia, an applicable and cost-effective system for acquiring pervasive traffic information based on WSNs. Recently, Bottero et al. [10] have installed and tested a WSN traffic monitoring system in the area of a logistic platform at the Turin's freight village in Italy. In SG, sensors can be embedded in metering devices, placed at both end-points and in the transport network, to monitor and control the energy usage and/or the waste in real time both locally and remotely. They help operators and consumers to manage their energy usage efficiently, reducing their energy bills and optimize delivery networks.

1.2 Motivation

WSNs differ from conventional distributed systems in many aspects. First, sensor nodes are only equipped with a limited power source (e.g., batteries and solar cells). Therefore, sensor nodes are easy to fail or have unpredictable operations. In addition, sensor nodes can be deployed in large numbers over a large geographic area or even in hostile or harsh environment. Therefore, replacement of sensors that have run out of energy is complex or sometimes impossible.

Second, current applications of WSNs are used for a single purpose solely and assuming that their operation parameters are fixed before de-

ployment. However, application requirements may change over time. For example, to deploy a new thief detection application in the smart house in Figure 1.2a, it would be more efficient if the new application can use existing devices in the house such as occupancy sensors rather than requiring the installation of new ones. In addition, it is convenient to have only one system which manages all the devices in the house. This helps the user to monitor and handle all the devices in the house easily. Therefore, a WSN should be able to support various types of applications or adapt to the changes in application requirements. In other words, it is expected that multiple applications will be executed concurrently over a single wireless sensor network [11] [12].

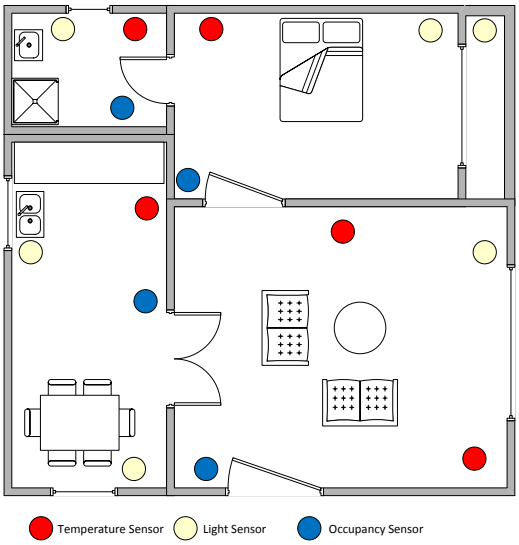
Third, there has been a lot of research conducted in both hardware and networking technologies for WSNs. WSNs can use different wireless node platforms (e.g., MICAz, TelosB, IRIS, Imote2) and different network protocols (e.g., MAC protocols [13], topology control protocols [14] [15] and routing schemes [16] [17]). Therefore, to manage a huge amount of heterogeneous sensor nodes and their data in WSNs is extremely complex.

Fourth, there will be more and more deployed WSNs to meet different needs in future. It is possible and expected that these networks can cooperate to support each other to improve the performance or the quality of offered services. For example in Figure 1.2b, based on the traffic information from the traffic monitoring sensor network, the pollution sensor network can predict the future pollution and change the collecting data rate correspondingly.

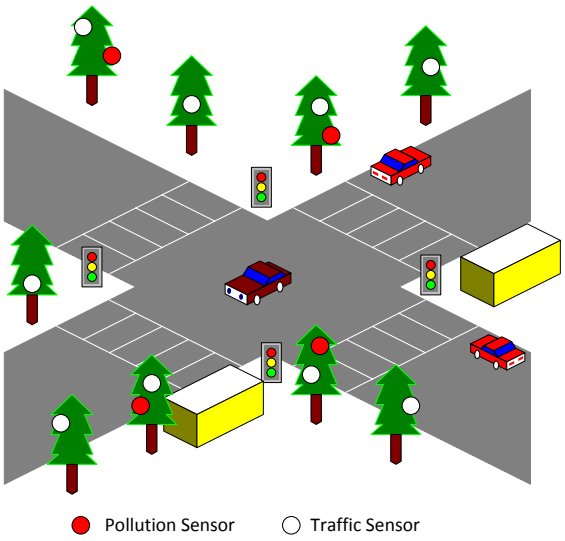
These existing limitations in WSNs make management problems on sensor nodes quite difficult to address. In order to outline the requirements of WSN management systems, we discuss some following management scenarios:

Fault or Misbehavior

There are many faults or misbehaviors which can happen in WSNs. In the following we will discuss two examples.



(a)



(b)

Figure 1.2: WSNs Applications: (a) Smart House, (b) Road Network

Scenario A: The electric bill is wrong due to no receiving energy consumption reading caused by errors on node, e.g., battery depletion or sensor broken, or errors on the data delivery path, e.g., network partition or network congestion.

Scenario B: The electric bill is wrong due to receiving incorrect energy consumption readings caused by external attacks, misbehaviors of metering sensor, or errors in network protocols.

These two scenarios provide some different requirements of WSN management systems:

- First of all, a WSN management system must be able to determine what has caused the faults. This requires management tasks such as monitoring and fault tracking.

- To avoid unexpected effects when a fault occurs, a WSN management system needs to support fault predictability. In other words, it should be able to detect a fault before it occurs by analyzing and validating data including sensing data, network operation logs, etc.

- Due to the existence of inevitable faults (e.g., in hardware, in software components, and in network links etc.), a WSN management system needs to detect these faults promptly and reconfigure network operations to ensure the accuracy of the provided service.

Integration of new sensor nodes or new applications

During the network's lifetime, there might be the need for deploying new sensor nodes or new applications to replace broken ones, to extend the network, to improve the network performance or to meet new users' requirements. The following are some examples of this situation:

Scenario A: A company wants to deploy a particular security application in its office, which is located in a smart building. This application includes some kinds of sensors such as camera sensors, motion sensors and occupancy sensors to capture any unauthorized activity. There might

be also some WSN applications with different types of sensor nodes deployed in the smart building such as the lighting system, the air conditioning system, and the alarm system. Taking advantage of existing resources in the building can reduce the deployment cost of the new user's security application. For example, it can utilize existing occupancy sensors of the lighting systems instead of deploying new ones.

Scenario B: In this scenario, sensor nodes powered by batteries are replaced by ones powered by solar energy in case sensor nodes are located in areas where sunlight is abundant such as green fields or roads. New energy harvesting sensor nodes can eliminate the inconvenience of replacing batteries, and also prolong WSN operational lifetime.

Scenario C: The deployment of a network may include several phases. In each phase, some new sensor nodes may be added to the network.

The management issues that arise in the above scenarios are as follows:

- Sensor nodes should be able to support multiple applications which can be owned by multiple users. A WSN management system needs to be able to allocate resources among applications, and also to ensure the privacy of each user.
- A WSN management system needs to have a power management mechanism to manage the harvested energy at harvesting sensor nodes. This mechanism should be able to cooperate with the resource allocation function to align the workload with the energy availability at sensor nodes.
- The integration of new sensor nodes or new applications can require a code update process. Due to the large number of nodes in WSNs, a manual update is inefficient. Therefore, a WSN management system should have a remote configuration function.
- In order to ensure the compatibility between old sensor nodes and new ones, a WSN management system needs to update the network operations in which new sensor nodes can take part in, such as routing or allocating network resources for applications.

Quality of Service of WSNs

Due to the variety of applications in WSNs, the quality of service (QoS) of WSNs varies greatly from application to application. For example, one of the QoS factors is the accuracy. In WSNs which provide information about the physical environment, the accuracy is measured by the discrepancy between the real world value and the provided results. However, in WSNs which are used to decide how to control actuators, the accuracy is measured by the discrepancy between the correct decision and the taken one. Moreover, different QoS factors such as delay and network lifetime may conflict by nature. Two scenarios are introduced to illustrate the conflict among QoS factors.

Scenario A: In the fire detection system in a smart building, important events such as high temperature and smoke occurrence need to be detected promptly. It requires a high data collecting rate which results in larger energy consumption, more network congestion and higher delays.

Scenario B: There are two WSN applications deployed on a road. The first application is used to detect the traffic congestion. The second one is to detect vehicles that cross a stop line while a red traffic light is on. There is a traffic congestion on the road. To keep live reports, camera sensors need to transmit information of the congestion (e.g., the vehicle density, the length of congestion, the beginning and the end of congestion) with a high rate to the sink, which affects the data traffic of the red light application. Information of some cars which violate traffic rules may be lost.

From the above scenarios, a WSN management system must consider the following requirements in order to ensure the required QoS:

- The WSN management system should define a QoS model for each application to identify the desired trade-off among QoS factors. It should also identify key QoS factors, if any, that influence the efficiency of the application. For example, the accuracy and the delay are more important for fire detection compared to other factors.

- The WSN management system should have a mechanism to monitor the QoS of running services to detect if the QoS of a service is

met.

- When multiple applications are executed concurrently in a single WSN, the WSN management system should combine the QoS models of all applications and generate a global QoS model to find the general trade-off in case the required QoS of all running applications can not be guaranteed.

Collaboration among WSNs

As mentioned above, there are multiple WSNs deployed to support different applications in Smart Cities. However, the WSNs operate independently and belong to different authorities. It would be efficient if the WSNs can cooperate to provide higher services or to improve the network performance. Some scenarios of the collaboration among WSNs are described as follows:

Scenario A: A driver wants to find a parking place around a tourist attracting area. The smart parking WSN and the traffic monitoring WSN can collaborate to guide the driver to the most suitable empty parking place without trouble of traffic congestion.

Scenario B: Based on collected information from the traffic monitoring WSN, the pollution monitoring WSN can adjust its data collecting rate correspondingly (e.g., the more traffic the higher rate). This helps the pollution monitoring WSN to keep up-to-date information of pollution while optimizing energy consumption.

To support the collaboration among WSNs, there are new management requirements which a WSN management system needs to take into account.

- The WSN management system should be able to analyze and validate data or requests received from other WSNs. Then, it should reallocate network resources to perform received requests.

- The WSN management system needs to monitor and evaluate the effects of the collaboration with other WSNs on the network perfor-

mance. It should be also able to use information of similar previous collaboration requests in handling the current request.

1.3 Objectives

Meeting the above mentioned management requirements requires an autonomic management platform. The autonomic concept is inspired by the autonomic function of the human central nervous system which helps human adapt to the changes of environment. This system regulates the body processes such as breathing, heat rate automatically without a person's conscious effort. The term autonomic computing was coined by IBM in [18]. The key feature of autonomic computing is self-* behavior which allows a system to manage itself without direct human intervention when its scale and complexity grows. The self-* behavior were described as follows.

- **Self-healing**: discover and repair potential problems to ensure that the system runs smoothly.
- **Self-protection**: identify threats quickly and take protective actions. Sensors feed data to a protection center, for auditing, and action taking against various threats.
- **Self-configuration**: install and set up applications/patches/updates automatically, verify compliance with the specified service levels, optimize configuration of applications using adaptive algorithms.
- **Self-optimization**: constantly monitor predefined system goals and performance levels to ensure that all systems are running at optimum levels.

The main objective of this work is to demonstrate that applying autonomic principles in the design of management systems for WSNs will bring a dramatic improvement to the scalability, the stability, the efficiency, the reliability and the flexibility of WSNs. In order to achieve the research goal, the following tasks are set:

- To design an autonomic management platform that enables sensor nodes to self organize in an efficient way with respect to the WSN restrictions and various applications' requirements. First, the desired features of such a platform need to be addressed. Second, its network architecture needs to be designed to support a variety of sensor networks from small to large scale, and from single to shared sensor networks.

- To design management data models and management protocols that enable the self configuration and the self optimization in both handling network operations and running user applications.

- To evaluate the scalability, the stability, the efficiency, the reliability and the flexibility of the proposed platform in specific and challenging scenarios.

1.4 Methodology

For any network management system or framework, a key challenge is evaluation. Most of the existing works in management for WSNs were not evaluated or only through small scale simulation or device emulation with a few sensor devices. We believe that the management system for WSNs should be evaluated on both simulation and testbed for various networks with different typologies, network sizes and different application requirements.

In this thesis, we firstly use SENSE [19], an independent simulator to give initial evaluation. Then, we deploy our approaches on TOSSIM [20], a simulation tool that included in TinyOS [21], a popular operating system for WSNs. Finally, we evaluate on testbeds including many TelosB motes.

1.4.1 Simulation

Since sensor nodes have limited computing power and memory sizes, it is difficult to verify the operation of algorithms on a sensor node. In ad-

dition, the number of sensor nodes can be large, and sensor nodes can be deployed in harsh environments. Therefore, to test and debug the management system in motes is extremely difficult and requires much effort. For example, if the designed algorithm causes a crash memory error, the mote stops working. It is difficult to know which part of the algorithms causing the error. For the early stages of the development of management systems for WSNs, using simulator can help to reduce the time debugging simple errors. The simulator also allows to evaluate the performance of the proposed system easily and quickly.

SENSE Simulator

SENSE [19] is a simulator designed with the primary goal is to bring the extensibility, reusability, and scalability for WSNs simulation. In order to enable the fully extensibility of network simulation architecture, SENSE proposed a component-port model where a component can be replaced by new one if they have compatible interfaces and inheritance not required. Moreover it also allow users to extend the simulation engines as needed. The reusability comes from the removal of interdependency between models e.g., the same module can be used at different simulations and the use of C++ template. In addition, SENSE provides the parallelization as an option to the users.

The simplicity of SENSE enables to build all components in the management systems from scratch. Although this takes much effort, possible errors in developing algorithms are detected easier since the problems of external libraries have been excluded.

TOSSIM

One of operating systems for sensor nodes is TinyOS [21]. TinyOS is based on nesC, a component oriented language. TinyOS provides a set of important services and abstractions, such as sensing, communication, storage, and timer and is compatible with many platforms. Moreover, it also provides some examples to help user better understand and easier develop sensor network applications. Building an application in TinyOS

is simply the definition of how to combine components. By using ”split-phase” approach, the hardware/software boundary in TinyOS is flexible.

TOSSIM [20] is a WSN simulator which is useful for assessing and evaluating application code in TinyOS operating system and Micaz mote based environments. It provides varying degrees of scalability, realism, and detail for understanding the behavior of sensor networks. One advantage of using TOSSIM is the developing system in TOSSIM can be easily deployed on testbeds which support TinyOS. Therefore, TOSSIM can be used as a validation step before testing on testbed.

1.4.2 Testbed

Simulation tools can provide a lot of information about the feasibility and the efficiency of the new management system. However, they can not cover all problems that can occur when deploying sensor networks for real business applications. For example, some data packets can be lost due to the occurrence of obstacles in the deployment area or a broken forwarding node.

TelosB

TelosB is a mote designed for easy experimentations and low-power operations. It has a TI MSP430 16-bit microcontroller with 10 kB of RAM and 48 kB of flash program memory. Its radio, a TI CC2420 which follows the IEEE 802.15.4 standard, can send up to 250 kbps. In terms of power, the radio dominates the system: on a pair of AA batteries, a Telos can have the radio on for about four days. Lasting longer than four days requires keeping the node in a deep sleep state most of the time, waking only when necessary, and sleeping as soon as possible.

1.5 Research Contributions

This thesis comprises scientific contributions to the WSN management research area. In summary, following contributions are made:

- We address management requirements of WSNs through presenting some key management scenarios in the Smart Cities context, such as intelligent transportation systems, smart grids and smart buildings. The limited resources and heterogeneous characteristics of WSNs pose new challenges in network management, which include the fault detection and diagnosis in the presence of various faults, the fault prediction and prevention to avoid replacing and repairing a large number of sensor nodes, how to ensure the efficiency of management processes under uncertain topology, and the resource allocation among heterogeneous sensor nodes for multiple applications. Then, we present in detail the desired features for a WSN management system such as lightweight, self-detection, self-configuration, sharing infrastructure, service monitoring, plug and play, context awareness and interoperability. We discuss advantages and disadvantages of centralized and distributed management approaches and the benefit of the multilevel management schema.

- We describe a DIstributed Self-Organizing Network management framework (DISON) that provides an autonomous management mechanism to allow sensor nodes to self configure and adapt to the changes in application requirements, resources and network state. In DISON, sensor nodes perform management tasks at different level according to their resources. Sensor nodes with limited resources can exploit their local knowledge to reconfigure their operations and provide management information for other nodes. In case of nodes that have more resources (e.g., more powerful battery or larger memory), more complex management tasks can be performed by them. One example is they can coordinate application tasks among a set of nearby nodes. Another example is they collect management data from a set of nodes to detect if there is any problem and reconfigure the operation of these nodes if needed. Sink nodes or base stations which are normally connected to large power suppliers and have strong computing capacity are responsible for global management tasks such as topology management and disseminat-

ing new code to support new management tasks. To support this architecture, we defined a new protocol stack and three data models. The Context and Policy models help sensor nodes to detect changes which they have to react, and to determine which action needs to be executed correspondingly. The Task model is used in handling multiple concurrent tasks.

- We describe some management mechanisms and protocols to manage WSNs based on the proposed platform. The context detection and policy based reasoning mechanism is used to detect context and trigger the corresponding policy. We have designed and implemented a role election protocol to assign management roles to sensor nodes according to their resources. We also describe an all in one acknowledgment mechanism to improve the accuracy of the exchange management data process. A resource allocation protocol is designed to coordinate multiple tasks from multiple authorities to sensor nodes in an efficient way to optimize the network resource usage and to prolong the network lifetime. The last mechanism, a local task scheduler, is responsible for scheduling all application task executions on sensor nodes in time to accumulate idle time duration in the processor schedule.

- Finally, we show by means of two case studies how the proposed management platform can be used in two data collection applications. The first application is very popular in WSNs, where a single authority (sink) broadcasts non concurrent requests to the networks. By contrast, there are multiple requests from multiple authorities in the second application. It is easy to see that the requirements of the second application are more complicated than the first one. This demonstrate the suitability and openness of the DISON platform to support applications from simplicity to complexity. Both case studies are evaluated in simulation tools and testbed.

The contributions generated by this thesis have been published in:

- DISON: A Self-organizing Network Management Framework for Wireless Sensor Networks, ADHOCNETS 2012, Paris, France.
- Design of a Generic Management System for Wireless Sensor Networks, Ad Hoc Networks Journal, Elsevier.
- Managing Heterogeneous WSNs in Smart Cities: Challenges and Requirements, submitted.
- A Management Schema for Shared Sensor Networks, submitted.

1.6 Thesis Structure

The remainder of this thesis is structured as follows. We begin in Chapter 2 with a discussion of sensor network management systems and approaches. In Chapter 3, we present the architecture and basic components. Some management data models are defined in Chapter 4. Chapter 5 introduces some management protocols and mechanisms that are the enabling technology for this platform. As proof of concept, we demonstrate the feasibility of the proposed platform by developing two different data collection applications, one in a single sensor network and another in a shared sensor network, in Chapter 6. Chapter 7 summarizes the thesis and draws the future trends.

Chapter 2

SENSOR NETWORK MANAGEMENT

This chapter is comprised of four parts. First, we will briefly outline the WSN management concepts, then its objectives, and the unique challenges WSN management brings. Next we will discuss some early works in management systems for WSNs. Finally, we will examine the key enabling technologies for network management in WSNs.

2.1 Objectives

Based on the management requirements described previously, a WSN management system can be defined as: *A management system for WSNs is an **autonomic** system that keeps the network and the services that the network provides up and running smoothly with as **little human intervention** as possible, and consumes as **little resources and energy** as possible. It predicts potential problems, performs operations to avoid or locate them, and self-configures or suggests solutions to solve them. It also allows adjusting network operations and reprogramming nodes remotely. Finally, it supports allocating resources to the services offered by the network.*

The overall of network management for WSNs is to examine the way in that network resources are being used, and provide the necessary infor-

mation for adjusting the operation of network so it optimizes the network usage and prolongs the network lifetime. According to the above definition, the detail objectives of network management in WSNs are:

- Managing network resources and services: monitor, control, update and report the status of sensor nodes and offering services.
- Reliable services: provide network with the high quality of service. Management systems for WSNs should detect, diagnose, fix, predict and prevent faults and errors.
- Little administrative effort: WSN network management systems should enable sensor nodes to self manage as much as possible.
- Prolong network lifetime: allocate network resources. Network management should have the ability to choose a set of sensor nodes to offer a required service. It should also arrange and coordinate network resources to serve multiple services from different authorities.
- Over-the-air update: WSN network management systems should be able to reconfigure or reprogram network remotely.

These objectives are accomplished through basic management activities, each of that must be provided in an effective WSN management system:

- **Monitoring** is one of the most important management functions. It is responsible for collecting the information required by the management system to monitor the running status of the network, including the network topology, the remaining energy of nodes in the network, and the quality of provided services, among others.
- **Resource Allocation** is necessary when multiple tasks, from different applications, run simultaneously in the same node and network. The resource allocation protocol is responsible for assigning network resources to different applications in order to ensure the quality of provided services while prolonging the network lifetime.

- **Fault Management** is responsible for the analysis of the causes and the search of solutions when a fault occurs.

- **Configuration** is used to reconfigure node's operation and update new code.

2.2 Challenges in Management for WSNs

Due to the scarcity nature of resources and the variety of applications, a WSN management system needs to cope with many challenges which are described in detail as follows:

- **Fault detection and diagnosis in the presence of various faults.** In WSNs, faults happen more frequently than other communication networks for many reasons. Firstly, sensor nodes have very limited resources. They are mainly equipped with a small power source (e.g., 2 AA batteries), which only allows them to be continuously active for hundreds hours of operation. In addition, batteries may be defective, hence, shortening node's lifetime. Therefore, sensor nodes are prone to failure due to the depletion of batteries. Secondly, WSNs can be deployed in various environments such as houses, buildings, roads, and rivers. There are a lot of factors which can make sensor nodes or network links fail temporarily or permanently. For examples, nature disasters or traffic accidents can break connections or destroy sensor nodes in one area. Thirdly, WSNs can have a large number of sensor nodes in a small area. In other words, the congestion may occur frequently since multiple nodes may want to transmit packets simultaneously, which leads to packet losses. The characteristics of WSNs make management more challenging. As has been discussed, there are multiple different factors that can cause problems and failures in WSNs. Therefore, figuring out exactly and promptly their causes is extremely challenging for the WSN management system.

- **Fault prediction and prevention is must-have.** WSNs might be deployed in remote, unattended, or hostile environments, which makes difficult, expensive or sometimes impossible to replace or repair broken sensor nodes. In those conditions, potential faults should be identified or predicted before they occur. How sensor nodes are able to predict potential faults and find solutions to prevent them is still an open challenge.

- **Ensuring the efficiency of management processes under uncertain topology.** Depending on the application, the sensor network topology can be random or pre-determined. For example, in a smart house or a smart building, the location of the sensor nodes is specified. However, in forest fire detection systems, sensor nodes are deployed randomly. Moreover, after the deployment, there may exist a lot of factors that affect the WSN topology, including node faults, different wake up cycles or node movement. For example, node faults might result in broken links and the loss of network connectivity, or sensor nodes can wake up at different periods due to mis-configured or faulty network protocols. In some applications, the sink or sensor nodes are placed on movable objects such as a patient, a vehicle or an animal, resulting in a changing network topology. When the network topology is uncertain, keeping up-to-date network information is more costly since the WSN management system has to monitor frequently the network state. Moreover, management data could be also lost due to a change of the management data forwarding paths. It would result in the degradation of the efficiency of the WSN management system.

- **Resource allocation among heterogeneous sensor nodes for multiple applications.** Traditional WSNs are designed to support a single application that belongs to a single user. However, with the rapid development of MEMS technology, there are more differentiated types of sensor devices with different energy capacity and functionality. This results in the emergence of heterogeneous WSNs that consist of several different types of sensor nodes and,

in addition, each sensor node may support multiple applications. For example, in a smart business building, the owner may deploy a WSN which supports multiple applications, including temperature and humidity monitoring, structure health monitoring and security alarms. Using a single network to interconnect all nodes can reduce the deployment and maintenance costs since each node can run several applications. However, that situation also raises new challenges for the network management, such as how to allocate the network resources to different applications, how nodes collect and transmit measured data of different applications from different nodes to the sink efficiently, and how to keep the energy consumption as low as possible. In addition, recent advances in technologies enable sensor nodes to collect and use energy from the environment [22], for example, light, differences in temperature, or linear motion instead of batteries. However, the availability of the harvested energy varies with time in a non deterministic manner. For example, the energy extracted from a solar panel depends on the maximum solar radiation and varies during a day. In addition, different nodes will have different harvesting opportunities. For example, sensor nodes placed at abundant sunlight areas can gather more energy than ones in shaded areas. Therefore, it is difficult to allocate tasks to the harvesting nodes since they do not have a stable energy source.

2.3 Early Works towards Management Systems for WSNs

In traditional networks, Simple Network Management Protocol (SNMP) [23] is a standard protocol for managing networks. In SNMP, a manager station collects information from network agents in network elements. The structure of the management data in SNMP are described in Management Information Base (MIB). There are several reasons that make SNMP popular. First, it can manage a large range of devices. Second, it is a very flexible and extensible management protocol. Third, it is also proved to

be good under poor network conditions. However, it requires transferring large amounts of management data between the manager and agents. This can potentially result in high energy and bandwidth consumption.

Ad Hoc Network Management Protocol ANMP [24] is an extended SNMP for wireless ad hoc networks, however, it can be used with certain types of WSNs. ANMP uses a hierarchical clustering mechanism for data collection to reduce the number of messages exchanged between the manager and agents (mobile nodes). In ANMP, the cluster head is responsible for collecting data from agents and forward them to the network manager. The nodes serving as cluster head change over time to adapt to node movements.

One of the earliest management systems for WSNs is MANNA [25]. It provides a general framework for policy-based management of WSNs. In MANNA, management services are executed by a set of functions. These functions are designed with a specific implementation for individual objectives in consideration of the unique features of WSNs. In order to provide the desired management services, MANNA defines policies that include conditions obtained from WSN models that should be satisfied, for which specific functions are executed. The relationship among WSN models are defined in MIB which are updated frequently to adapt to network changes promptly. It is critical to determine the right time to query for management information and the right frequency for obtaining management information to ensure the accuracy of collected information while keeping low energy consumption.

In [26], Younis et al. divided the network into multiple clusters in which each cluster has a gateway node that organizes and manages network operations based on application requirements and the available energy in sensor nodes. However, their approach mainly focus on finding data relay route and arbitrating medium access.

Song et al. [27] presents another management system for WSNs based on Universal Plug and Play (UPnP) [28], the standard service discovery protocol for network management. It consists of three main components: control point, BOSS, and non-UPnP sensor nodes. The control point is a powerful device to support UPnP protocol. BOSS (Bridge Of the Sen-

sorS) is an UPnP agent, which is implemented in the base station and lies between the UPnP controllers and the non-UPnP sensor nodes to be managed. It contains the services of each sensor to provide them with a control point. It interprets and transfers messages between the sensor network and the control point.

Toller and Culer [29] proposed SNMS, a management system for WSNs which provides two mechanisms: query-based health data collection and persistent event logging. The query based health data collection mechanism allows users to collect the network data indicated in physical parameters to monitor the network health. The event logging mechanism allows nodes to store log events and send them to the users when they are requested. An improvement of SNMS based on Remote Procedure Call (RPC) mechanism is proposed in [30].

WinMS (Wireless Sensor Network Management System) [31] is an adaptive policy-based management system for WSNs. In WinMS, network states are monitored continuously to collect management data. When management parameters exceeds predefined thresholds, WinMS executes management tasks to reconfigure the network. In WinMS, individual sensor nodes can perform management functions locally based on the network state of their neighbors. The base station works as a central manager which stores and analyzes the global state of the network to detect interesting events and execute management maintenance.

2.4 Key Technologies

We have identified a set of enabling technologies that are used for WSN network management. We will first examine policy based management approaches. Then agent based approaches are discussed. Finally, we will present some middleware approaches.

2.4.1 Policy based management approaches

Policy-based management has emerged as a promising solution for the management of large-scale and heterogeneous networks. In policy based network management approaches, policies are defined as rules that govern the states and behaviors of the network system. Such policies are device independent and human friendly. Policies would be automatically updated to adapt to the changes of network state. Such automation is an essential requirement for large networks with frequent changes such as WSNs. However, one disadvantage of policy based management approaches is its functional rigidity, that is, we can not add new management services to the system, unlike agent based management approaches.

Some early policy based management systems for WSNs are MANNA [25] and WinMS [31]. In MANNA and WinMS, policies specify management functions that should be executed if certain network conditions are met. Both of them use the central server to analyze network states and executes corrective and preventive management actions according to predefined policies.

Other policy based management approaches for WSNs are introduced in [32], [33]. In [32], Cha et al. proposed a hierarchical framework in which the base station is responsible for interpreting high level management policies and distributing them to sensor nodes. These policies are then applied locally on each sensor node if its state matches. High level policies are defined in XML schema. They are distributed and interpreted to low level policy at sensor nodes. Le et al. [33] propose SRM, a hierarchical management architecture and policy-based network management paradigm for WSNs. There are three levels of policies in SRM: node level, cluster level, and base station level. At the node level, management policies consist of rules that require less resources and can be perform locally. The cluster level contains management polices that control the reliability of the cluster. The highest level, the base station level, include polices that control the entire network.

Zhu et al. [34] proposed Finger, an efficient policy based management system. Finger supports interpretation and enforcement of both obliga-

tion and authorization policies on all sensor nodes. Obligation policies are event-condition-action rules that perform an action in response to an event. Authorization policies define what resources or services a subject can access on a target sensor. Each sensor node has a Policy Decision Point (PDP) and a Policy Enforcement Point (PEP). The PDP is responsible for interpreting policies and making decisions. The PEP enforces the policy that is the result of PDP decision.

Policy based management systems can be also found in [35][36] [37] [38]. Matthys and Joosen [37] propose a policy driven middleware architecture to manage distributed sensor applications in a network infrastructure that consists of several sensor networks to offer services for different types of users. The proposed architecture supports two types of policies: functional policies, which are high level management goals and non functional policies, which are concrete goals that can be executed. Bourdenas and Sloman [38] describe the Starfish framework for specifying and dynamically managing policies in sensor nodes. It uses Finger2 which evolved from the original Finger system [34] to interpreter and enforce policies.

Policies based management approaches are also used to manage some particular areas in WSNs. For example, policy based energy management systems are presented in [39] [40]. Bourdenas et al. [41] present a framework for autonomic task allocation in sensor networks based on Starfish [38]. Waterman et al. [42] have described the Peloton OS architecture that allows to distribute resource allocations to meet some desired policies. Misra and Jain [43] design a policy to activate the optimum number of sensor nodes such that the application fidelity is not affected based on the concepts of Markov Decision Processes (MDP).

2.4.2 Agent based management approaches

In agent based management approaches, a mobile agent is defined as is a section of code that can distribute management tasks to be executed on nodes locally and returns the resulting data to the central manager [25]. The local processing can help to reduce the network bandwidth to

the central manager server. However, some special nodes are required to act as agents and perform management tasks. These nodes should be placed intelligently to cover all the nodes in the network. In addition, the manager has to wait for the agent to visit the node in order to retrieve its status. This can cause delay. Some common examples of agent based management systems for WSNs are presented in [25] [44] [45] [46] [47].

Erdogan et al. [44] propose sectoral sweepers (SS) for managing a wireless sensor network. Each region of the network has a sectoral sweeper. The sectoral sweeper allows the central server to enable or disable tasks on nodes within a certain network region.

Agilla [45] is a mobile agent middleware designed to support self-adaptive applications in WSNs. It enables applications to be locally and autonomously self-adaptive by integrating the mobile agent and the tuple space programming models. Each sensor node maintains a tuple space that contains a set of predefined descriptors about that node. These tuple spaces can be accessed remotely. Each sensor node can be monitored by multiple agents. An agent can be cloned or moved across nodes. Agilla was designed for TinyOS operating system [21].

There are several agent systems based on Java which are introduced in [46] [48] [47]. MASPOT [46] is a brand new mobile agent system natively designed for the Sun SPOTs (Sun Small Programmable Object Technology) sensor devices [49]. Muldoon et al. [48] adopt Agent Factory Micro Edition (AFME), an intelligent agent framework for ubiquitous devices to sensor nodes. In [47] Haghighi and Cliff propose a novel middleware solution, which runs on Java (SE and ME) programming platforms for easy task distribution and data gathering integrated in a modulated architecture that supports the serving of multiple concurrent applications, dynamic reprogramming, good scalability, and multiple operational paradigms.

2.4.3 Middleware approaches

Middleware approaches add an additional logic layers within the firmware of motes in order to implement management services. These approaches

provide a runtime environment that can support and coordinate multiple applications. They also provide mechanisms that optimize the system resources usage. In [50], Heinzelman et al. describe MILAN, a middleware that allows applications to specify their quality needs and adjusts the network operations to prolong network lifetime while still meeting these required quality needs.

TinyDB [51] provides users a tool to query the network using SQL languages. It collects and transmits sensing data from motes in the environment to the sink. Impala [52] is a middleware architecture that enables application modularity, adaptivity, and repair-ability in wireless sensor networks. It allows software updates to be received via the node's wireless transceiver and to be applied to the running system dynamically.

Mires [53] is a publish/subscribe middleware where the publish/subscribe service acts as a bridge between the local application and the communication components in a sensor node. Each node advertises topics that can provide. The user application receives these topics and selects the desired topics to be monitored. After this, nodes are able to publish the collected data of interest. Another public/subscribe middleware is presented in [54]. The middleware proposed in [54] provides application specific communication channels and an approach to transform incoming sensor data to the desired data representation.

As mentioned previously, Agilla [45] is a mobile agent middleware that facilitates the user application deployment process. The RUNES middleware [55] is a component-based programming model where units of functionality and deployment are encapsulated in components. These components interact with each other through interfaces. RUNES supports dynamic reconfiguration that allows to upload and offload components and code dynamically.

Shah and Kumar [56] have proposed DReL, a middleware framework that provides mechanisms and data structures to allow support of applications with different QoS requirements and optimization goal based on reinforcement learning and utility theory. In DRel, sensor nodes can decide whether to host an application task based on their capabilities and the utility of performing that task before. Ganz et al. [57] describe a mid-

dleware architecture that uses context information of sensors to supply a plug-and-play gateway and resource management framework for heterogeneous sensor networks.

There are several middleware systems that are proposed to support IoT applications recently. For example, SNPS [58] is an OSGi [59] based middleware that enable sensor nodes to be used and composed over the Internet in a simple and standardized way. Another example is MobIoT [60]. MobIoT is a service-oriented middleware that enables large-scale mobile participatory sensing. A new device can only register its service if it can provide new information that is not covered by the set of registered devices.

Chapter 3

A DISTRIBUTED SELF ORGANIZING NETWORK MANAGEMENT PLATFORM

This chapter presents DISON (DIstributed Self Organizing Network), a generic management platform for Wireless Sensor Networks (WSNs). The first objective of the DISON framework is to allow sensor nodes to adapt autonomously to changes in application requirements and network resources. The second objective is to provide a framework that enables both the developer and the administrator of WSNs to choose what management functions to perform, what conditions should trigger sensor nodes' adaption, and how to adapt in both the development and run-time stages.

This chapter is further structured as follows: section 3.1 presents the design goals of DISON, section 3.2 describes the multilevel management architecture of DISON, and section 3.3 presents the new protocol stack of sensor nodes designed for supporting DISON.

3.1 Design Goals

DISON is designed to enable the self management in sensor networks. We now list design goals for DISON.

Lightweight

Since sensor nodes have limited resources, a management system for WSNs should be as lightweight as possible. The management functions and the management process should only occupy a small memory size. There should be a trade-off between the network traffic generated by a management process and the benefit derived from it.

Self-detection

There are a variety of faults in WSNs. Simple faults which are caused by hardware error or battery depletion should be detected locally by every sensor node. A couple of simple faults from different nodes can lead to a complex fault (e.g., network congestion or network partition). Complex faults can have a high probability to cause a degradation on the network performance. Therefore, sensor nodes should be able to collaborate to detect complex faults from simple faults.

Self-configuration

Operation of sensor nodes should be optimized and able to adapt autonomously to the changes in resources and application requirements to prolong the network lifetime and prevent possible faults. For example, sensor nodes in the same sensing area can collect and transmit sensed data alternately. When a sensor node detects a fault, it should notify other nodes. Depending on the importance of the fault, sensor nodes should be able to adjust their operations to reduce negative effects caused by that fault. For example, if the sensing component of a sensor node is broken, it can have a more important role in the forwarding path since it does not

need to collect data from the environment. Therefore, other nodes can change their routing table to use that node as the forwarding node.

Sharing infrastructure

The deployment and maintenance of large WSNs with thousands of nodes require a high cost and huge effort. In case many WSNs are deployed in the same area, it would be efficient if they share their resources to support multiple applications from the different authorities. This is clearly seen in the two following examples:

- **Single application**. In a smart building, both the lighting system and the security system use occupancy sensors in rooms and corridors. In the lighting system, occupancy sensors are used to turn on/off the light depending on the presence of persons in the room. In the security system, they are used to start monitoring. Much of the same area is covered by both systems.

- **Single authority**. In a smart city, there are several organizations (e.g., police, highway agency, and local city authorities) that need to deploy their own camera networks on roads. However, these networks can cover the same areas and therefore, they may generate redundant information.

Therefore, it would be beneficial to have a single infrastructure supporting multiple applications. The shared infrastructure can include many different types of sensor nodes, in which some nodes support multiple applications. The management system of such an infrastructure should be able to allocate resources among applications to optimize the network performance. As the infrastructure can be shared by multiple authorities, it needs an access classification mechanism that assigns different privileges to different authorities to ensure the privacy.

Service Monitoring

The QoS of running services in WSNs should be monitored and evaluated periodically to detect whether it meets the predefined requirements. A management system for WSNs should also provide detailed information about the availability of the running services.

Context aware

As mentioned above, a WSN should support multiple applications. Since each application has different requirements in terms of network resources, and both the application requirements and the network resources change over time, the network behavior should adapt to optimize its performance. During the network lifetime, there might be some situations that can be predicted before they happen. For example, there are more customers at commercial centers during weekends than weekdays. Then, to offer customers a comfortable shopping environment, the commercial centers' smart systems (e.g., lighting and air conditioning) may increase the operating power and the operating frequency autonomously every weekend. In such cases, a management system for WSNs should be able to use information of the handled changes to process the current ones if they are similar. For example, the adjustment of the operating power and the operating frequency of the smart systems last weekend can be applied to the current weekend if the number of guests and the outside air temperature are similar.

Plug and Play

Due to the heterogeneity of WSNs, management functions should be as independent as possible from hardware, network protocols and user applications. The same management function should be able to work with different applications, different operating systems, different hardware and different network protocols. This would help to reduce the developing cost. In order to achieve this feature, management functions should be parameterizable and configurable. This allows to interface easily with

other network protocols, hardware functions and different applications. In order to optimize the memory usage, management functions should be only added to a node when they are needed, and therefore, they should be able to be added or removed easily.

Interoperability

In order to enable the collaboration among different WSNs, a management system for WSNs should support interoperability. It means the WSN management system is able to communicate and exchange data with ones of other WSNs. Data standards and public interfaces should be defined and unified among the authorities of WSNs to facilitate the collaboration.

3.2 Network Architecture

One of most important aspects of the design of a network management system is its architecture. In this section, we describe the network architecture of the DISON platform.

3.2.1 Background

There are many kinds of network management architectures for WSNs. We can divide them into three groups: centralized architectures, distributed architectures and hybrid architectures.

Centralized approach

A centralized management server that processes the management data and take management decisions may be the best option for small networks. This central management server collects information from all sensor nodes and controls the entire WSN operation [61][62]. Due to its abundant resources and the global knowledge of the WSN, it can perform complex management tasks and provide accurate management decisions.

Complex management tasks are actions that require a high amount of resources and global knowledge of the network. For instance, controlling the network topology is a complex task. However, for large WSNs, it is difficult and costly to keep the management data from all the nodes in the network up to date. Firstly, sensor nodes cannot send management data frequently to the central server due to the high communication overheads of the multi-hop forwarding. Secondly, transmitting management data frequently to the central server increases the traffic load of the nodes close to the sink, which can cause network congestion and lead to high packet losses.

Distributed approach

Distributed management approaches are more suitable than centralized ones in large scale networks. Management decisions are taken by multiple manager stations [25][32]. Each manager station controls part of the network (i.e., a group of nodes), and can cooperate with other manager stations if needed. However, the main disadvantage of the distributed approach is that manager stations do not have a global view of the network, as they only know the state of their respective subnetwork. Therefore, although their management decisions can be effective for their local subnetwork, they can affect negatively the operation of the overall network. For example, some nodes are turned off by a manager station in its subnetwork to optimize the resource usage. If those nodes happen to be the only connections to the rest of the network, the whole network will be severely affected. The cooperation among manager stations can improve somewhat this issue, but it may imply high overheads. For example, two manager stations can cooperate to decide which nodes are going to sleep without affecting the network connectivity. However, if the WSN has a lot of subnetworks, the number of management packets exchanged among manager stations will be high. Besides that, not all manager stations have rich power sources or strong processing capabilities, which means that the number of management functions or the complexity of management functions at those stations is limited.

Hybrid solution

Both centralized and distributed approaches have advantages and disadvantages. To take advantages of both approaches, a hybrid management architecture could be designed for WSNs. In a simple term, a hybrid management architecture consists of both centralized management server and manager stations to perform management functions based on the complexity and the cost required by these functions. These are the following advantages of a hybrid management architecture:

- **Reliable.** It can detect, handle, and isolate faults locally without affecting the functioning of the rest of the network. In can also provide accurate management decisions due to the existence of the centralized management server.

- **Scalable.** It is easy to increase the size of the network by adding new sensor nodes, without affecting the current network operation.

- **Flexible.** According to the changes in application requirements, the network topology, and the network resources, sensor nodes can have different management roles. For example, when the remaining battery of a manager station is low, one of its neighbors can become a new manager station to ensure the execution of management tasks in that area.

- **Effective.** Manager stations can be selected based on their resources or their roles in network (e.g., cluster head or parent node in routing tree). Therefore, the delay of handling management decisions and management overhead can be minimized.

Although the hybrid solution have many advantages, the design of a hybrid management architecture is complex. It requires to have an efficient algorithm to choose the manager stations. However, there are a lot of clustering algorithms developed for WSNs [63] which can be used to select manager stations, developed for WSNs. Another disadvantage of

the hybrid solution is the management overhead. Exchanging management data can cause high traffic and energy consumption. Due to the lack of resources in WSNs, a hybrid management architecture should be designed to ensure the trade-off between the management overhead and the efficiency of the management system. The number of exchanging management data should be minimized while it still ensures the accuracy of management decisions.

3.2.2 Multilevel management schema

Since we wish to be able to support a variety of networks, a multilevel management schema is designed for DISON as depicted in Figure 3.1. There are three levels of management in DISON. The lowest level is the self management that is a must for every sensor nodes. Self management is the ability of the node to deal with all that can happen with itself during the lifetime including the degradation of the battery, broken hardware, the lack of resources, and lost connections with its neighbors. Self management ability aims to place the sensor node in control of its life and enable it to be self adaptive.

The second management level is the group management. The network is divided into groups of nodes. Each group has one manager while others are the members. The manager node is responsible for helping its member nodes to handle management situations that they can not do by themselves. It has to ensure that the resources of its members are utilized efficiently. Constant monitoring by the manager helps to ensure that all members are working and any fault is detected promptly. Due to the resource scarcity of sensor nodes, the frequency of the constant monitoring could not be too small. It is because the small frequency results in the high traffic of management packets, hence, consuming more energy. Besides, there might be some cases that require a manager node to have the up to date information of its members. Therefore, a manager node should be able to request its members to provide information when it needs including the battery level, the sensing capabilities, the number of neighbor nodes, and the number of running application tasks.

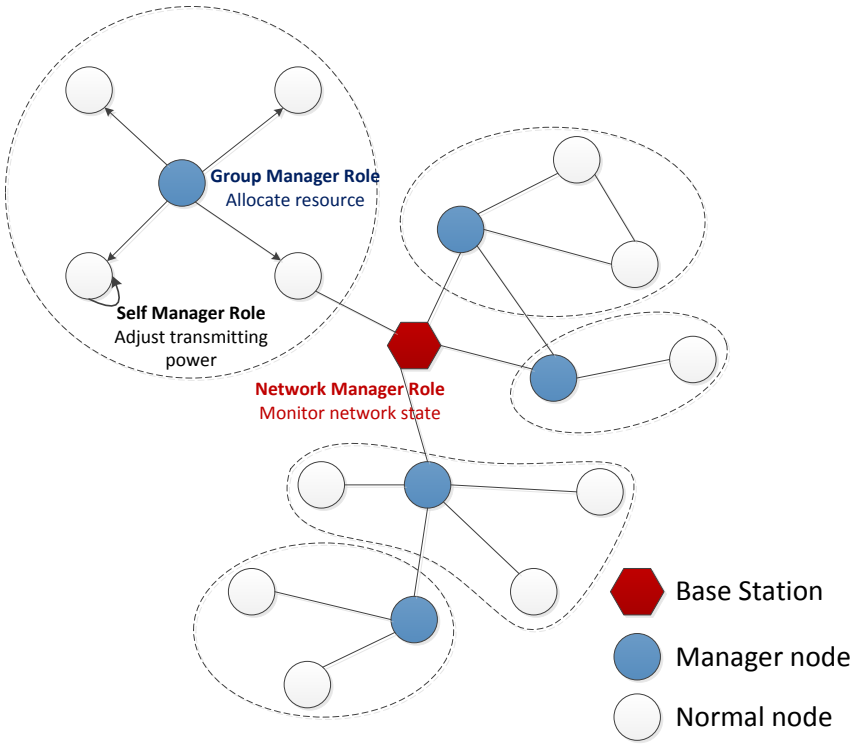


Figure 3.1: Multilevel management schema

There are several approaches to divide the network into groups and select manager nodes. One approach is using central servers (base stations) to choose which node is the manager node of a certain group of nodes based on the network topology. This approach is efficient for networks which have a fixed topology and a small number of nodes. Another approach is the use of clustering algorithms [64][65]. The chosen cluster heads, which are elected by using a clustering algorithm, are suitable to role as manager nodes. Since the clustering algorithms do not make any assumptions about the presence of infrastructure, they are useful for networks that do not have a fixed and stable topology, a common case in

wireless sensor networks.

The number of member nodes in each group is variable. It depends on several factors. One factor is the capability of the manager node. Because the manager node needs to store the information of its members, the larger number of member nodes is the larger memory space in the manager node is occupied. It also takes more time and more computing operation to access and process the information from the large number of member nodes. The other factor is the approach that divides the network into groups and chooses manager nodes for each group. In the central server approach, the network administrator can decide the number of nodes in each group. However, in clustering approaches this number is normally random and different to ones of other groups. Network density and network connectivity also affect the number of member nodes. Network areas that have different network density and network connectivity can be divided into groups with different sizes.

The last and the highest management level is the network management. Due to the limited resources, sensor nodes can not perform management tasks that require a powerful processor, a large memory space and a big energy source such as collecting network topology, maintaining state of all nodes, and monitoring the quality of all provided services in network. Therefore, the network management is assigned to sink nodes or base stations.

In order to support the multilevel management schema, DISON defines three management roles corresponding to three levels of management. The first management role is SELF_MANAGER_ROLE, corresponding to the self management level. It contains management functions that executed on every sensor nodes such as configuring node's operation (e.g., adjust the transmitting power and turn on/off the radio antenna or sensors), asking support from its manager node to validate an application task, and scheduling multiple concurrent tasks. The second role, GROUP_MANAGER_ROLE is assigned to manager nodes. It includes management functions that monitor and coordinate network resources and network operations of its members. For example, choosing only few active nodes to perform a required service may reduce the

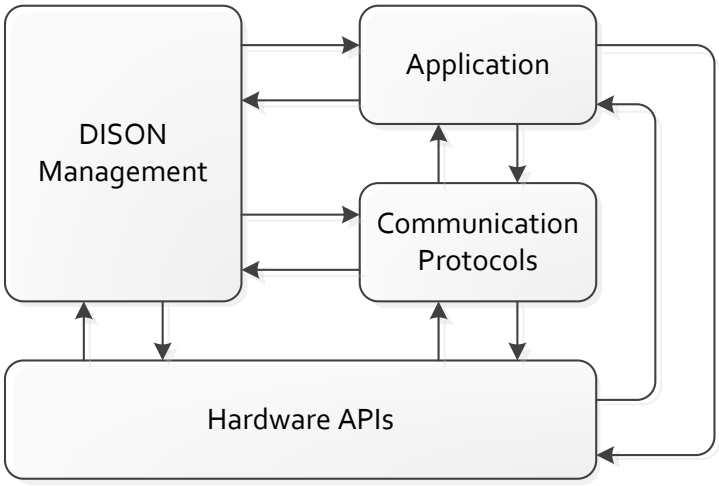


Figure 3.2: The DISON protocol stack

power consumption and prolong the network lifetime. The last role, NETWORK_MANAGER_ROLE, provides global management functions such as storing the network topology, monitoring the network state, merging or simplifying the application requirements from users before passing them to the network.

3.3 Node Architecture

In order to allow DISON to work independently to other network protocols and user applications, we propose a new protocol stack for sensor nodes. It includes 4 components: Hardware APIs, Communication Protocols, Application and DISON Management.

3.3.1 Protocol stack overview

The protocol stack designed for DISON framework is shown in Figure 3.2. The components of this protocol stack can be summarized as follows:

- **Hardware APIs** provides access interfaces to the controller, communication devices, storage resources, sensors and the power supply of the sensor node. Examples of these interfaces are switch on/off the node, read/write the memory, switch on/off the radio. It also perform management functions that process raw data received from the hardware (e.g., converting voltage reading to the remaining battery level, trigger alarms when the battery is low, or dropping invalid sensor reading data) and establish the acknowledge mechanism for data transport.

- **Communication Protocols** carries all network protocols that use to transmit data packets from the source to the destination. Examples are MAC, routing, location and time synchronization protocols.

- **Application** is responsible for collecting and aggregating sensing data. It handles user requests including application tasks and management tasks. It schedules and coordinate tasks at runtime on sensor nodes. It is also responsible for allocating network resources among members in groups of sensor nodes.

- **DISON Management** is the core components of the DISON framework. It provides management functions to access and modify management data including node's state, neighbors' information, link connectivity, running task's information, and members' information (in case of manager nodes). It is also responsible for grouping nodes and selecting manager nodes. It provides management mechanisms to monitor, detects problems, find out the reasons, trigger alarms, and suggests the solutions.

These components interact with each other through public interfaces. This enables the independence among components. The change of one

component does not affect other ones. For example, the resource allocation protocol in the **Application** component can optimize its operation depending on the application requirements to increase its efficiency. However, there are no changes needed in other components that use resource allocation functions.

To ensure the independence of DISON with respect the application and network protocols, management functions in the DISON components in each sensor node interact with the same ones in another node. For example, messages sent by the resource allocation protocol in the **Application** component can only be processed by the corresponding one at the receiver node.

3.3.2 Structure of components

We describe here in detail the structure of the Hardware APIs, Application and DISON Management components in the DISON protocol stack. In this thesis, the Communication component is out of scope. We use existing communication protocols to transmit data in the network.

DISON Management Component

As mentioned above, the DISON Management Component is responsible for providing the basic functions of the DISON platform. Figure 3.3 depicts the structure of the DISON Management Component. It includes five main parts.

- **Management Data** stores all information that uses in management processes. There are four main types of management data: Context, Policy, and Task. The Context data is used to indicate which information can affect node's operations. The Policy data is used to determine which action needs to be executed when a predefined context occurs. The Task data stores information of tasks provided by the sensor node itself or its members if it is manager node. The Node data stores information of nodes including node's state,

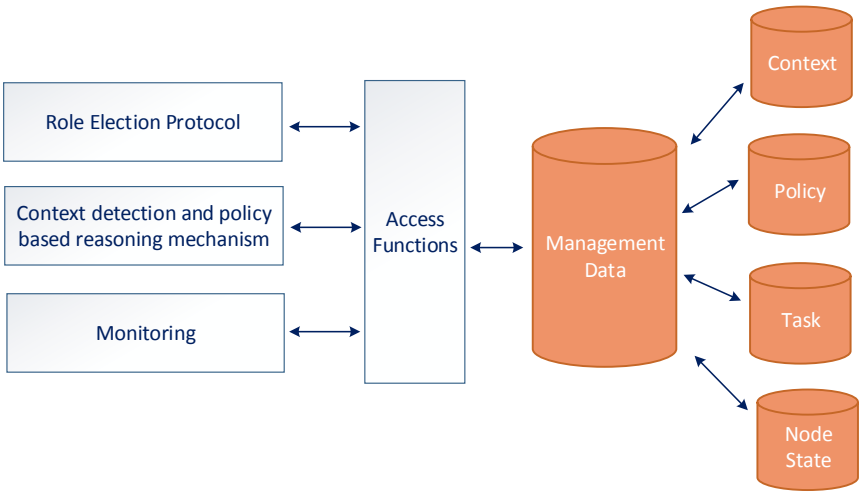


Figure 3.3: DISON Management Component Structure

neighbors’ information, link connectivity, and members’ information (in case of manager nodes).

- **Access Functions** provides management functions that add, update, access and remove management data. Most of these functions support public interfaces to enable other components to access management data.

- **Monitoring** is responsible for collecting the information required by the management system to monitor the running status of the network, including the network topology, the remaining energy of nodes in the network, and the quality of provided services, among others. Depending on the management role, the Monitoring functions can be different. For example, normal sensor nodes only monitor their state while manager nodes monitor both their state and the state of nodes in their group.

- **Context detection and policy based reasoning mechanism** is used

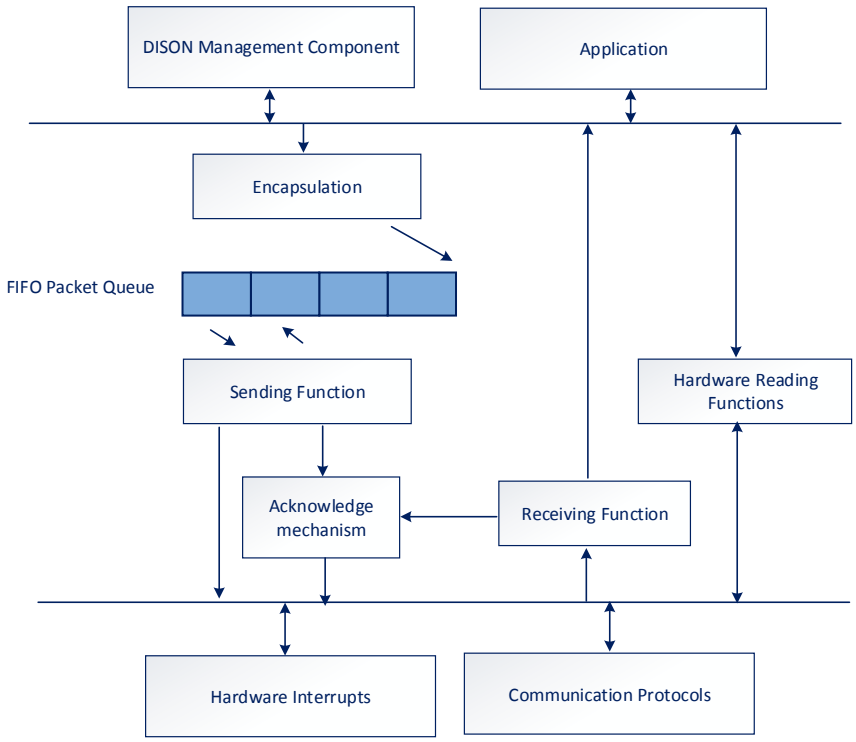


Figure 3.4: Hardware APIs Component Structure

to find out when a node needs to change its configuration. It works with Context and Policy data.

- **Role Election protocol** divides network into groups of nodes and selects which nodes are manager nodes. It also allows member nodes to leave the group when they want.

Hardware APIs Component

Figure 3.4 depicts the structure of the Hardware APIs component. It contains four parts.

- **Hardware Reading Functions** interacts with Hardware Interrupts to read and standardize node's state. For example, it reads the voltage of the battery (12 bits number) and converts it to a percentage battery level (8 bits number). It helps to reduce the memory storage. It also provides functions to reconfigure node operations. For example, it can change the duty cycle or turn on/off the radio control.

- **FIFO Packet Queue** is used to store packets before passing through Communication Protocols. Since there are multiple sources of packets in DISON platform (e.g., election protocol and resource allocation protocol), it is essential to have a packet queue to avoid packet losses when the radio is busy. DISON uses the First In First Out structure (FIFO) to make the operations with the queue simple.

- **Encapsulation** is used to add the management packet header before pushing to the FIFO Packet Queue. If the FIFO Packet Queue is full, it signals a fail notice.

- **Sending Function** is responsible for sending all the packets in the FIFO Packet Queue. In case a sender requires to have the confirmation from the receiver for the sending packet, Sending Function passes the packet through Acknowledgment mechanism. Otherwise, the packet is passed directly to the Communication Protocols.

- **Receiving Function** is executed when a packet is received from the Communication Protocols. If the packet is not an ACK, it is passed up to DISON Management or Application Components. Otherwise, the Acknowledgment mechanism is triggered to process the packet.

- **Acknowledgment mechanism** enables a sender to know that its packet is received successfully by all the expected receivers. In DISON, we support 1 hop ACK for all kind of communications including broadcast, multicast, and unicast.

Application Component

The last component we present in this section is the Application Component. The structure of the Application Component is illustrated in Figure 3.4. There are four independent parts.

- Sensing Data Collection is used to collect data from multiple sensors. It drops sensing data that does not meet the application requirements.
- Resource Allocation is responsible for deciding whether or not the application service or task is host by the sensor network or sensor nodes. It checks if the requested task is new and it has enough resources to provide the requested task. In case a sensor node belongs to a group, it can ask the support from its manager to make decision whether or not to offer the requested task. In case of manager nodes, they can support or request its members to provide a task.
- Task Scheduler provides the ability to schedule the launch of multiple tasks so that the node's resources are used efficiently.
- Data Aggregation combines data from multiple tasks to reduce the number of small sent packets.

3.4 Discussion

Many network management architectures for WSNs have been presented to solve management problems mentioned in the Chapter 1. Ruiz et al. [25] describe MANNA, a generic architecture for managing WSNs. In MANNA, management services are distributed to some manager nodes and agents, for which it defines a specific mechanism to distribute efficiently these management agents. The delay when performing a management task is also an issue in large scale networks since manager nodes need to wait agents visit the nodes. In our framework, the simple management tasks are embedded in all sensor nodes while complex management

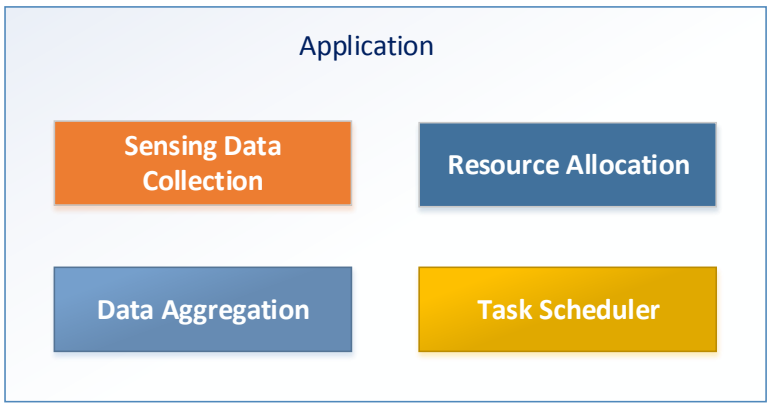


Figure 3.5: Application Component Structure

tasks are assigned to manager nodes which are easily setup by clustering approaches. Thus, management tasks in a network area can be performed easily and quickly. This feature allows our framework to adapt to large scale networks.

Some policy based management approaches are introduced in [32], [33]. In [32], Cha et al. proposed a hierarchical framework in which the base station is responsible for interpreting high level management policies and distributing them to sensor nodes. These policies are then applied locally on each sensor node if its state matches. Therefore, their approach requires the base station to maintain an up-to-date global view of the network which is not always feasible, especially in large scale networks. In our framework, management tasks are distributed locally in each node and each region (or each group of nodes). Since management data is mainly exchanged among nodes and their closer manager nodes, our proposal reduces the management overhead. Le et al. [33] proposed three level management policies to distribute management tasks to the base station, cluster heads and sensor nodes. However, their work only provides and evaluates the management system for data reliability management at the node level. In our approach, we discussed in detail the role of manager

nodes in allocating network resources in sets of nodes.

In [26], the authors divided the network into multiple clusters in which each cluster has a gateway node that organizes and manages network operations based on application requirements and the available energy in sensor nodes. However, their approach mainly focus on finding data relay route and arbitrating medium access. A hybrid management solution is presented in [31]. WinMS [31] allows individual sensor nodes to perform management functions locally based on the network state of their neighbors. In addition, the base station works as a central manager to store, analyze the global state of the network and execute management maintenance operations if it detects any event. Thus, WinMS has the same drawback as [32] when the number of nodes increases.

Reinforcement learning techniques are also used to support autonomous and adaptive management in WSNs such as in [56]. Shah and Kumar [56] have proposed a middle-ware framework in which sensor nodes can decide whether to host an application task based on their capabilities and the utility of performing that task before. The ongoing project TinyCubus [66] proposes a cross layer framework that allows systems to adapt to the evolution of the network and application requirements. They also developed algorithms to assign the network role (e.g., SOURCE, AGGREGATOR, CLUSTERHEAD etc.) and distribute code updates to nodes according to their roles. Other policy-based management systems are also presented in [35][36]. However, none of them provide a detailed model to represent data, analyze it and support at the same time both application requirements and management tasks.

There is some research that focus on supporting multiple applications in WSNs in recent days. SenShare [12] proposes a hardware abstraction layer on each sensor node to allow multiple concurrent applications to use the node's hardware resources. In [11], Majeed and Zia provided a switching mechanism to activate or inactivate a set of nodes depending on which application has to be executed.

Agilla [45] is a mobile agent middleware designed to support self-adaptive applications in WSNs. It enables applications to be locally and autonomously self-adaptive by integrating the mobile agent and the tuple

space programming models. A programming framework for WSNs has been proposed in [67] to provide abstractions for object-centric, ambient-aware, service oriented sensor network applications. More comprehensive reviews and the comparisons of wireless sensor network programming models can be found in [68].

The design idea of DISON framework is similar to Di-Sec and M-Core, two security frameworks proposed in [69] and [70]. All of them are aiming at providing a generic, easily extended, modular and flexible framework for WSNs. However, Di-Sec and M-Core focus on solving security problems while DISON works on management issues. Another work which has some similar features with DISON has been proposed in [71]. In [71], Prinsloo et al. designed a service oriented architecture to allow WSNs applications to be developed faster and easier by reusing generic components.

In [72] and [73], Octopus and NetViewer, two monitoring, visualization and configuring tools for WSNs have been designed. Octopus [72] enables the user to view the network topology and reconfigure network parameters (e.g., change wake up duty cycle or application request). Many of the graphical interface design choices in DISON stem from Octopus. NetViewer allows to view the information of packets from different applications based on their XML packet formats.

Different to other works in management systems for WSNs, DISON aims at providing a generic and self-adaptive management framework. Management functions not only allow WSNs to adapt to the changes on both the network resources and application requirements, but also they are self-adaptive. In DISON, management functions can be added, removed or optimized according to the user requirements. Moreover, DISON also allows to allocate network resources locally, to a group of sensor nodes. This helps to optimize the resource usage while keeping the management overhead low, especially in large scale and high density networks.

Chapter 4

MANAGEMENT DATA MODELS

The DISON framework supports three data models: Context, Policy and Task. We define context as the knowledge indicating which information can affect node's operations. The policy is used to determine which action needs to be executed when a predefined context occurs. Context and Policy models can be represented in XML language [74] at the network manager. However, it could not be applied in normal sensor nodes because they have very limited resources. The last model, Task Model, is used to represent application tasks that are received or provided by sensor nodes. In this chapter, we discuss how to represent three models in a sensor node.

4.1 Context Model

The concept of context and context aware systems was early introduced in [75]. There is a lot of research that developed context awareness protocols and frameworks for WSNs such as in [57] [76]. However, there is still no comprehensive studies on the generic and formal representation of contexts for WSNs, in particular, for sensor nodes.

As defined above, a context indicates which information can result

in a reconfiguration action. For instance, a reconfiguration action can be changing the transmit radio power or denying an application task, etc... One example of a context is "The remaining battery of the node is low".

There are a lot of factors that affect the node's operation. However, we can divide them into three following groups:

- **Node Resource.** This group includes information about node's local resource such as the sensing capabilities, the residual energy, the memory usage.
- **Application Requirements.** This group includes information about user's requirements such as desired query, expected event.
- **Network State.** This group includes information such as neighbor information, loss link rate.

Each context in DISON framework includes 5 elements:

- CONTEXT_ID. CONTEXT_ID is the unique identifier of a context in the network. The intent of CONTEXT_ID is to allow the network administrator to add or update context easily. Because the sensor nodes and sink nodes have different capabilities, we propose to use 1 byte for CONTEXT_ID in sensor nodes, and 2 bytes for CONTEXT_ID at sink nodes. To ensure the CONTEXT_ID is unique, identifiers from 0 to 127 at the sink nodes are reserved for contexts which are used in sensor nodes.
- INF_TYPE describes the source of the information. In our framework, we use 2 bits to represent three above groups of information sources.
- INF_ID represents the identifier of each specific information such as sensing capability, RF power, current voltage, query request. The size of INF_ID depends on the INF_TYPE. For example, we propose to use 3 bits to represent INF_ID for the Node Resource group because there is few information in the node can be collected to use in the management process.

- OPERATOR indicates which logic operator ($=$, $>$, $<$, etc.) is used. Since the computing capability of sensor nodes is limited, 4 bits is enough to represent this field.

- VALUE represents the threshold to fire a management decision. If the measures metric is higher than this threshold, a management decision is requested. We propose to use 4 bytes for this field. The unit of the VALUE field depends on the INF_ID. In order to minimize the memory storage, the threshold should be standardized to a small number. For example, the voltage reading value of TelosB and MicaZ is a 12 bits number. If we know the type of equipped batteries (e.g., AA 2300mAh or AA 2800mAh), we can know the maximum voltage which can be measured in each node. We can assume that the maximum voltage is corresponding to the case in which the battery is fully charged. From that, we can divide the current voltage reading value by the maximum voltage to know the percentage of the remaining battery.

Some examples of context are shown in Table 4.1. The first three contexts are related to the remaining battery. The unit of the VALUE field for the remaining battery is percentage. For example, the first context means that "The remaining battery of the node is under 20%".

CONTEXT_ID	INF TYPE	INF ID	OPERATOR	VALUE
1	NODE_RES	BATTERY_LEVEL	<=	20
2	NODE_RES	BATTERY_LEVEL	>	20
3	NODE_RES	BATTERY_LEVEL	<=	40
4	NODE_RES	SENSING_CAP	=	1
5	NW_STATE	NB_SEND_FAIL	>=	5
6	NW_STATE	RADIO_ERR	=	1
7	NW_STATE	NB_LAST_RECV_PERIOD	>=	1
8	APP_REQ	MAX_CONCURRENT_TASKS	=	3
9	APP_REQ	TASK_ACCEPT	=	1
10	APP_REQ	TASK_ACCEPT	=	0

Table 4.1: Examples of Context

POLICY_ID	SET OF CONTEXTS	SET OF ACTIONS	PRIORITY
1	1	(ALARM, BATTERY_LOW)	1
2	2 AND 3	(APPROVE_TASK, FALSE) AND (ADJUST_RADIO, REDUCE)	1
3	9	(APPROVE_TASK, TRUE)	2

Table 4.2: Examples of Policy

4.2 Policy Model

The second important data model is the policy model. We form a rule based policy as the following formula:

POLICY_ID [SET OF CONTEXTS] [SET OF ACTIONS] PRIORITY

The left hand side of a policy includes identifiers of contexts (CONTEXT_ID) which are combined by conditional elements such as 'AND', 'OR', 'NOT', etc. The right hand side is the set of actions to be executed when the policy is applied. Each action is represented by a couple (ActionID, Parameters) where ActionID is the identifier of the action (e.g., reconfiguring networks, route discovery) and Parameters is the list of the parameters used by the action. Each policy is identified by an unique identifier POLICY_ID. The PRIORITY field is used to select which task is triggered when there are two policies that conflict between them. The lower value that PRIORITY is, the higher priority the policy has. Some examples of policies are shown in Table 4.2. When the context 1 "Battery Level is under 20%" occurs, the policy 1 is triggered. Node executes the function ALARM to notice to its manager node that its battery is low. If the context 2, 3 and 9 occur concurrently, the policy 2 will be triggered while the policy 3 is ignored because the policy 2 has higher priority. Therefore, node denies the task and reduce the transmitting power.

4.3 Task Model

In shared sensor networks, application tasks arrive randomly. Each task has different requirements. Since the main goal of WSNs is collecting data when the user requests or a special event occurs, we use query language to model application tasks. TinyDB [51] presented an acquisitional query language for WSNs. One example of the query in TinyDB is as follows:

```
SELECT nodeid, sensingtype
FROM sensors
WHERE temp > 20
```

```
SAMPLE RATE 1s FOR 10s
```

The above query specifies each sensor node should report its own identifier (id), light and temperature readings every $1s$ during $10s$. In this thesis, we extend the query language of TinyDB to support the shared sensor networks by adding some new attributes including USERS, PRIORITY, CELL, and a bit code of sensing type.

```
SET USER userid
SET TASK localtask_id PRIORITY priority_value
SELECT nodeid, sensingtype s
FROM sensors
WHERE condition
CELL sensing_area
SAMPLE RATE r FOR p
```

The SET USER clause indicates who requests the query task. Each task has an localtaskid. The pair (userid, localtaskid) can be used as the global identifier of the task in the network. The PRIORITY attribute is used when a new task arrives while the number of running tasks in the network reaches its limit. If the priority of the new task is higher than the one of the running tasks, the network stops the task whose priority is lower and executes the new task. When any running task terminates, the network will restart the stopped task. The CELL attribute indicates the sensing area of a sensor node. If two sensor nodes have same sensing type and are placed in the same CELL, their collecting data should be similar. For example, the temperature sensor of the thermostat and the one in the air conditioner in one room should return the same temperature value. The CELL attribute can be used to query data in a specific area. In addition, sensor nodes that have same CELL can alternately provide an application task. This can help to prolong the lifetime of each node while meeting the application requirement.

We represent the sensing capabilities of sensor nodes as shown in the Table 4.3. Each sensor node can have a single or a composed sensing type. A composed sensing type means that the sensor node has several sensing types. For example, sensor node can have both temperature and light sensors. We used a limited number of bytes to represent both single

Sensor Type	Bits	Meaning
NO SENSOR	0000000	No sensor is enabled
TEMPERATURE	0000001	Temperature sensor is enabled
LIGHT	0000010	Light sensor is enabled
HUMIDITY	0000100	Humidity sensor is enabled
ACCELEROMETER	0001000	Accelerometer sensor is enabled
MAGNETOMETER	0010000	Magnetometer sensor is enabled
MICROPHONE	0100000	Microphone sensor is enabled
SOUNDER	1000000	Sounder sensor is enabled

Table 4.3: Sensing Capability Value

and composed sensing types where each bit position is corresponding to one type of sensor. The example in the Table 4.3 uses 1 byte to represent 7 different sensing types. This method is simple and fast with low memory cost since it can use bitwise operations such as AND, OR, XOR. For example, a sensor node only needs to perform AND operation between its sensing capability and the requested one to see if it can provide the requested service. However, this representation method has one drawback. The number of representing bytes can be large when the number of sensing types increases. Since each bit is corresponding to a single sensing type, assume that there is n sensing types, the number of representing bytes is $n/8$. In case there are 128 different sensing types, the number of representing bytes is 16 bytes. We use 1 byte to represent the sensing capabilities to support 8 different sensing types in our platform.

We now give some preliminary definitions. Let $\mathcal{T} = \{T_1, ..., T_n\}$ be the set of tasks on a sensor node. Each task $T_i \in \mathcal{T}$ has parameters s_i, r_i, p_i, c_i where s_i indicates what sensing type is needed, r_i is the sampling rate, p_i is the period of collecting sensing data, and c_i is the remaining data that needs to collect.

Definition 1. T_i is an active task if T_i is executed, that is, the sensor node

collects and transmits data to the sink as required in T_i. Let $\mathcal{A}$ is the set of active tasks, $\mathcal{A} \subset \mathcal{T}$.

Definition 2. A sensor node can provide T_i fully if it has all the requested sensors.

Definition 3. A sensor node can provide T_i partly if it has at least one requested sensor but does not have all the requested sensors.

Definition 4. A task with parameters (s, r, p, c) is duplicated if following conditions are satisfied:

- in case the role of the sensor node is SELF_MANAGER_ROLE, the task is duplicated if and only if:

$$\exists \mathcal{A}' \subset \mathcal{A} \mid s \subset \bigcup_{T_i \subset \mathcal{A}'} s_i \quad \text{and} \quad p/r < \min_{T_i \subset \mathcal{A}'} c_i \tag{4.1}$$

- in case the role of the sensor node is GROUP_MANAGER_ROLE, the task is duplicated if and only if:

$$\exists \mathcal{T}' \subset \mathcal{T} \mid s \subset \bigcup_{T_i \subset \mathcal{T}'} s_i \quad \text{and} \quad p/r < \min_{T_i \subset \mathcal{T}'} c_i \tag{4.2}$$

Chapter 5

MANAGEMENT PROTOCOLS AND MECHANISMS

Management protocols and mechanisms are the fundamental parts of a management system. This chapter describes several management protocols and mechanisms designed to use within the proposed platform. The context detection and policy based reasoning mechanism provides information that can trigger events at an appropriate time. In order to assign management roles in a network, we design a role election protocol to choose powerful sensor nodes as manager nodes. We describe a resource allocation (RS) mechanism to allocate resources among nodes in groups of adjacent nodes when a new application task arrives. Finally, we propose a local task scheduler mechanism to enable multiple application tasks to be executed concurrently.

5.1 Context Detection and Policy based Reasoning Mechanism

In this section, we discuss how a node knows when it has to change its configuration. In other words, we describe how to detect a context or how to find a policy to trigger. As described in chapter 4, there are three

sources of context information. Each source needs a different approach to collect the information. For example, the Node Resource group only needs to execute hardware functions to collect the remaining battery or the radio state. But the Network State or Application Requirements group may need to cooperate with other nodes to collect information such as the last time it received a packet from its neighbors or the response of its manager node to accept a task.

As soon as the information is collected successfully, the node replaces the corresponding information in the existing context with the collected one. If the logical expression returns true, that context occurs. One example is if the remaining battery is 18% , the expression $18 <= 20$ returns 1, that is, the context 1 occurs (Table 4.1). Another example, the last time the node A receives a packet from the neighbor B is at 30:10. Assuming that the current time at node A is 30:15. So the NB_LAST_RECV_PERIOD will be 5 minutes. Therefore, the context 7 occurs (Table 4.1).

When a context occurs, it is easy to find the matching policy by searching the CONTEXT_ID in the list of policies. In case the policy needs more contexts (e.g., in Table 4.2 the policy 2 needs both the context 2 and the context 3 but only context 2 occurs), the node adds the current context 2 to its buffer. When the context 3 occurs, the node checks its buffer and finds out that the policy 2 is satisfied. Then the node executes the functions which are indicated in the policy 2. If there is a new context which conflicts with the context in the buffer, the new one will be added to the buffer and the old one will be deleted. For example, when the context 1 occurs, the context 2 is deleted (Table 4.1).

When a policy is satisfied and triggered, the POLICY_ID is also added to the buffer. If there is a new satisfied policy, the node checks if there is any conflict with the active ones. The functions in the policy which has higher priority will be kept or executed while the functions in the another one will be canceled or ignored. For example, let us consider that the policy 3 in Table 4.2 is satisfied and that the node approves the task (e.g., the query task) by starting to collect data. After a period, the remaining battery is getting low and the policy 2 is satisfied. The node finds out that it conflicts with the policy 3. Since the policy 2 has higher priority than

the policy 3, the node cancels the collecting data process and adjusts its transmitting power.

5.2 A Role Election Protocol

The core idea of the DISON platform is the multilevel management schema. Therefore, organizing sensor nodes into groups or sub networks and choosing manager nodes are ones of the basic functions that need to be executed before any other management functions.

5.2.1 Background

As discussed in Chapter 3, there are two approaches to divide the network into groups and select manager nodes. One is using central servers. Another is using clustering algorithms. In the scope of this thesis, we only discuss the clustering approaches.

An extensive survey on clustering protocols proposed for WSNs is presented in [77]. In most of clustering protocols, cluster heads are selected randomly or based on a weight function [78] [65] [79]. The parameters used in the weight function can be the signal strength, residual energy, and the intra cluster communication cost.

In order to elect manager nodes, we have designed and implemented a simple manager election protocol based on [65]. We choose nodes which have high capability as manager nodes. The capability of a node has to include information about both local resource and network state. The reason is that manager nodes have to store and process management information of their members. If a manager node has not enough resources, it can not perform complex management tasks that require much resources or the accuracy of its management decisions can be affected. Similar, if a manager node has good link connections to its members, the exchange of management messages is easier and more reliable.

In order to enable the developer and the network administrator to change which information is used in the election process, we define a

general weight function that covers all possible information about a sensor node. In addition, we allow a sensor node to leave its current group if it receives a better joining group offer from other node.

5.2.2 Description of the protocol

We have defined the following equation to calculate the capability of a node:

$$A = \frac{\sum_{i=1}^{n} \gamma_i \cdot a_i}{\sum_{i=1}^{n} \gamma_i} \tag{5.1}$$

where a_i is a value that represents the state of a certain node's resource such as CPU speed, free memory storage, battery level, etc., that is normalized into range $[0, 1]$. The number n is the total number of different resources used to evaluate the node's capability. Each type of resource has a priority γ_i that shows its importance. The value i is the unique identifier of a resource. The users can configure the value of n and γ_i to adapt to the network state or the application requirements. For example, when the remaining battery of all nodes is high, the network connectivity is more important than the remaining battery. But when the remaining battery is getting low, it may become more important than to provide full-connectivity to non-interesting areas.

In the scope of this thesis, to calculate the capability A, we only use the remaining battery level, sign as a_1, and the number of neighbors, sign as a_2, defined as follows:

$$a_1 = \left\lfloor \frac{\text{Current Voltage}}{\text{Maximum Voltage}} \right\rfloor \qquad a_2 = \left\lfloor \frac{\text{Current Number of Neighbors}}{\text{Maximum Number of Neighbors}} \right\rfloor$$

The corresponding priorities are γ_1 and γ_2. Thus, the capability of a node is calculated as following:

$$A = \frac{\gamma_1 \cdot \left\lfloor \frac{\text{Current Voltage}}{\text{Maximum Voltage}} \right\rfloor + \gamma_2 \cdot \left\lfloor \frac{\text{Current Number of Neighbors}}{\text{Maximum Number of Neighbors}} \right\rfloor}{\gamma_1 + \gamma_2}$$

In our protocol, a sensor node can have three possible states: it can be a manger node, a member of a group or a free node. In the beginning, all nodes are free.

At initialization, every node collects information which is used to calculate the capability A. For example, every node broadcasts the HELLO message to its neighbors to announce its presence. They also read the remaining battery level from the hardware. Then, the node which has capability higher than a predefined threshold becomes a manager candidate and broadcasts the Role Offer (ROF) message. The ROF message includes the identifier and the capability A of the node that has sent it. The broadcast is limited to 1 hop.

When a node receives a ROF message from one of its neighbors, there are three possible situations. In the first situation, the node is a free node. If the capability A in the ROF message is higher than its capability, it accepts that neighbor as its manager. Then it sends a Role Confirm (ROC) message which includes its identifier back to its manager to notice its decision. In the second situation, the node belongs to a group. If its manger has smaller capability than the received one, it sends a Group Leave (GL) message to its current manager and a ROC message to the new manager node. In the third situation, the capability A in the ROF message is smaller than the one of the node or its manager, the node ignores the received ROF message.

There are also two possible situations when a node M receives a ROC message from a node S. First, if node M is a free node, it becomes a manager node and adds node S to its member database. If the number of node M's members reaches its limit, the manager node M sends a Role Denied (ROD) message to node S. In the second situation, node M is a member of a group. If node M has not seen any message from its manager for a long time, it sends a GL message to its old manager, and then sets

itself as the manager of the new group. Otherwise, it ignores the received ROC message.

If the manger node receives a GL message, it clears the information of the corresponding node from its members. In case a member node receives a RD message from its manager, it resets its state and becomes a free node.

The process is executed periodically to ensure the balance of the network resources since manager nodes consume more resources than normal nodes as they perform complex management tasks.

5.2.3 Summary

We present a role election protocol for organizing the sensor network into disjoint groups of nodes. In each group, a node is selected to be the manager node. The proposed protocol extends the weight function in existing clustering protocols to contain all possible information for the manager election. This weight function can be configurable by enabling/disabling the presence of a resource state value or changing the corresponding priorities. The proposed protocol also allows sensor nodes to be able to leave their group to join other group which has more powerful manager node or more activity. It means only active groups with powerful manager nodes can exist.

5.3 An All-in-one Acknowledgment Mechanism

As we know, the data transmission in the WSNs is more unreliable than it is in the traditional networks. In this section, we propose a simple and efficient acknowledgement mechanism for reliable management message exchange in DISON.

5.3.1 Background

In dense networks, there is a high probability of losing management packets due to the network collision, hence, leading to a decrease in perfor-

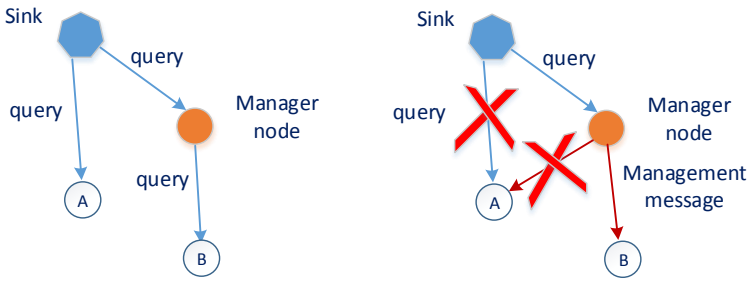


Figure 5.1: A loss management message scenario

mance of both the management framework and the network. For example, in Figure 5.1 the sink broadcasts a query to all nodes in the network. The manager node chooses node A or node B to collect data. Assume that it chooses node A. However, due to some reasons the management decision message sent to node A is lost. Node A did not receive the query from the sink either. So it did not know it has to collect data. Besides, node B receives the management decision message from the manager node and skips the requested query. Therefore, there is no node to support the requested query.

There are several works that provide data reliable mechanisms for WSNs. IEEE 802.15.4 standard [80] supports the ACK at MAC layer level to improve the transmission reliability. However, it is only available for unicast communication. A reliable broadcast mechanism is introduced in [81]. RBP [81] lets each node flood the received broadcast message only once. The retransmissions are performed when the number of ACKs seen by the node falls below a predetermined threshold. RBP uses information of node density and topology information to adjust the retransmission threshold and the number of retries. RBP is implemented between the MAC layer and the routing layer. Works in providing ACK mechanisms for multicast are still missing.

5.3.2 Description of the protocol

We have designed and implemented a new ACK mechanism that supports all three types of ACK including unicast ACK, multicast ACK and broadcast ACK. The proposed ACK mechanism is implemented as a module in the Hardware APIs Component. Currently, our ACK mechanism is only used in exchanging management messages. However, it is easily to apply to other kinds of network messages.

In our ACK mechanism, each time the sender sends a message packet, it can request the receiver to confirm that the packet is received by sending an ACK message when necessary according to the importance of message. For example, ACKs for HELLO messages are not necessary while using ACKs when sending ROF and ROC messages is essential.

We added a new field *noACK* to the common management message structure. We use 1 byte to represent this field. The meaning of the value of the field *noACK* is as follows:

- $noACK = 0$: The ACK mechanism is not required.
- $0 < noACK < 255$: The sender requires the receiver to send back an ACK. When the number of ACKs received at the sender is equal or larger than the one indicated in the *noACK* field, the message packet is considered as sent successfully. If after a period Δ_{ACK}, the sender has not received enough ACKs, it tries to send the message packet again. If the sender can not receive enough ACKs after MAX_RETRIES_TIMES, it considers that the message transmission has failed.
  - $noACK = 1 \Rightarrow$ Unicast ACK
  - $noACK = number\ of\ neighbors \Rightarrow$ Broadcast ACK
- $noACK = 255$: In this case, the sender only requires a subset of its neighbors to send back ACKs (multicast ACK). The next byte in the packet, the one after the $noACK$ field, indicates the number of neighbor nodes that need to send back an ACK, then the following bytes are the identification of these neighbor nodes. When the

receiver receives the multicast ACK request, it only sends back an ACK if it is in the list of nodes indicated in the message packet.

After sending a message that requests ACKs, the node keeps the sent message in the FIFO Packet Queue and sets a timer T_{ACK} to wait for ACKs. If the node receives enough ACKs before the timer T_{ACK} expires, the message is removed from the FIFO Packet Queue. Otherwise, the node retransmits the message. After MAX_RETRIES_TIMES, the message is removed from the FIFO Packet Queue.

5.3.3 Summary

We provide a simple ACK mechanism that uses only local density, i.e., the number of neighbor nodes to increase the reliability for one hop transmission. Depending on the kind of management messages, the appropriate type of ACK can be used.

5.4 Resource Allocation Protocol

In the previous chapters, we discussed the emergence of shared sensor networks as an integrated infrastructure for multiple applications, and the management challenges in such networks. In this section, we present a resource allocation protocol that efficiently allocates the network resources to different applications.

5.4.1 Background

An efficient resource allocation protocol is necessary when multiple tasks, from different applications, run simultaneously in the same node and network. The goal of a resource allocation protocol is to assign network resources to different applications in order to ensure the quality of the provided services while prolonging the network lifetime.

There are some noticeable solutions for allocating tasks in WSNs which have been proposed recently. A novel approach to adaptive re-

source allocation in single sensor networks, called Self-Organizing Resource Allocation (SORA) is presented in [82]. Each sensor node acts as a self-interested agent that selects actions to maximize profit, subject to energy limitations. Nodes self-schedule their local actions in response to feedback in the form of payments. This allows nodes to adapt to changing conditions and specialize their behavior according to physical location, routing topology, and energy reserves. Other works in resource allocations for single WSNs can be found in [83][42].

Adaptive Servilla [84] is a middleware that coordinates resources in heterogeneous WSNs subject to the energy efficiency and network dynamics. It treats services as the basic unit of software modularity so that they can be dynamically offered to different applications. Li et al. [85] proposes the Resource Allocation in Heterogeneous WSNs (SACHSEN) algorithm to allocate applications (including simultaneous arrivals) to sensor nodes based on the application utility which is decided by selected QoS requirements. SACHSEN supports both application QoS and network QoS. However, all of works only allow to allocate multiple tasks locally. Based on the multilevel management schema of DISON, we propose a resource allocation protocol that allocates network resources to different applications among adjacent nodes, where manager nodes support their members in making decision whether or not to provide an application task.

5.4.2 Protocol overview

The resource allocation protocol is used to decide whether or not the application service or task is host by the sensor network or sensor nodes. Fig. 5.2 illustrates the task registration process. Initially, the node is in the IDLE state. When a task arrives to a node from the user request, the node firstly verifies if it can support that task by checking its on going tasks list and its resources. In this stage, the node changes its state to VERIFYING. If the answer of the verifying process is yes, the task is executed and added to the on going task list. Concurrently, the state of the node transits to APPROVED. In case the node can not decide to ac-

cept the task by itself, it can ask for the support from its manager node by sending a Task Register (TREG) beacon which includes the information of the received task. Simultaneously, it starts a timer T_v and changes the state to REFERRING. When it receives the Task Response message (TREP) from the manager node, it accepts or denies the task depending on the answer. If the task is denied, the node state transits to SKIPPED. Otherwise, it changes the state to APPROVED.

In this function, a manager node is responsible for selecting which sensor nodes in its group are candidates to perform the request task so that the quality of service is not affected. After receiving TREG beacons from nodes in its group, the manager node analyzes the capabilities of each sensor node, and the requirements of the requested task. Based on this information, it makes a list of candidates and broadcast the TREP message including the list of candidates to the sensor nodes in its group.

Once a task is approved, the node adds the task to its ongoing task list. If there are more than one task on the list, then the node executes a multiple task optimization mechanism which reschedules the running time of the on-going tasks, aggregates the data generated from different tasks before transmitting them to the base station or generates a new common task which substitutes the new and old tasks. Some examples of this mechanism in case of multiple queries are presented in [86], [87]. Depending on application requirements, the developer can implement the appropriate multiple task optimization mechanisms to optimize the performance.

The resource allocation process enables sensor nodes to adapt to the changes in application requirements. Every time there is a new application task or a change in an old one the resource allocation process chooses which sensor nodes will be used to perform that task based on the current state of the network resources. This process is executed partially in each group of sensor nodes to reduce the management overhead.

5.4.3 Description of the protocol

We now present the detail of the resource allocation protocol. The task model used in this protocol is mentioned in Chapter 4. Since there might

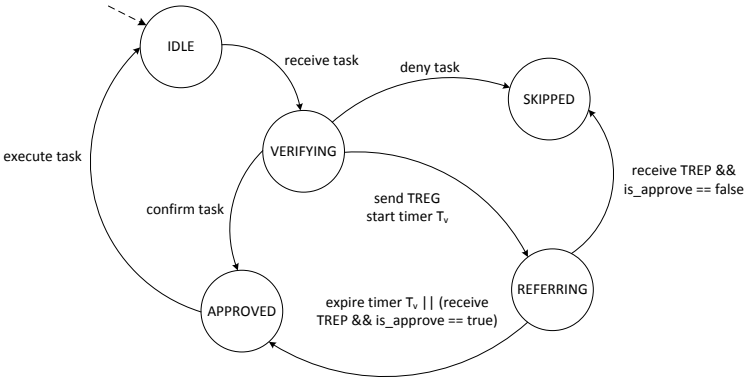


Figure 5.2: The state diagram of the task registration process

be multiple tasks which arrive at the same time, we use a simple FIFO queue, naming it as $DecisionQueue$ to store waiting decision tasks. The decision process for the task that arrives later is only started when the previous one has been completed. The size of the $DecisionQueue$ in manager nodes should be larger than in normal sensor nodes because manager nodes process not only its tasks but also tasks of its group members.

Each time the $DecisionQueue$ has a task to process, the sensor node verifies its sensing capabilities to see if it has enough requested sensors. In case a normal sensor node can provide the requested task fully or partly, it sends a TREG message to its manager node to ask for support, and then schedules the decision process for that task. In case of a manager node, it creates or updates the decision schedule. The interval of task decision timer in manager node should not be smaller than the one in normal sensor node; otherwise the response from the manager node might be lost. Let τ_s and τ_g be the interval of the task decision timer on normal sensor node and manager node respectively. To avoid missing a decision response from the manager node, $\tau_s > \tau_g$ is required.

When a manager node receives a TREG message from one of its members, it pushes the requested task to its $DecisionQueue$ if that task is new. When the task decision timer fires, if the normal sensor node has

not received the response from its manager node, it can accept or deny the requested task according its sensing capability. In case of a manager node, it chooses a subset of its member nodes which have more energy and less active tasks to provide the requested task. Then it broadcasts a TREP message that includes the identification of all nodes in that subset to its member nodes. As receiving the TREP message, if a sensor node belongs to the list indicated in TREP, it accepts the requested task.

The pseudo code of the resource allocation protocol is described in Algorithm 1.

Algorithm 1 Resource allocation algorithm

```
1:  function Sensor node: On arrival of a task T_i with parameters (t_i, s_i, r_i, p_i)
2:      isDuplicate ← checkDuplicate(t_i, s_i, r_i, p_i)
3:      if isDuplicate == TRUE then
4:          Ignore T_i
5:      else
6:          Add T_i to the list of tasks T
7:          Push T_i to the DecisionQueue
8:          if isWaitingDecision ≠ TRUE then
9:              isWaitingDecision ← TRUE
10:             Start processing task
11: function Task processing
12:     if DecisionQueue is empty then
13:         isWattingDecision ← FALSE
14:         return
15:     T_i ← DecisionQueue.head()
16:     if resources are fully or partly enough then
17:         if Role of node is SELF_MANAGER_ROLE then
18:             Send TREG message to manager node
19:             Schedule decision making at time τ_s
20:         if Role of node is GROUP_MANAGER_ROLE then
21:             Schedule decision making at time τ_g
22:     else
23:         Ignore T_i
24:         Remove T_i from DecisionQueue
```

The algorithm to choose a subset M of member nodes in a group to provide the new task T_i should be designed based on the characteristics of the network and application. For example, in a single task sensor network which is deployed in high density of nodes, M can be chosen based on the battery level and the link connection. However, in case of shared sensor networks with multiple tasks, the number of ongoing tasks should

Algorithm 2 Resource allocation algorithm (continued)

function Normal node: Receive a TREP message
  if Node is requested to run a task T_i **then**
    Add T_i to T in case T_i has not been seen before
    Run Task Scheduler
  if $T_i == DecisionQueue.head()$ **then**
    Stop decision making timer
    Remove T_i from $DecisionQueue$
    Process next task in $DecisionQueue$
function Manager node: Receive a TREG message which requests to register T_i
  if T_i has not been seen before **then**
    Add T_i to T
    Push T_i to the $DecisionQueue$
    if $isWaitingDecision \neq TRUE$ **then**
      $isWaitingDecision \leftarrow TRUE$
      Start processing task
function Decision making timer fired
  $T_i \leftarrow DecisionQueue.head()$
  if Role of node is SELF_MANAGER_ROLE **then**
    if resources are fully enough **then**
      Run Task Scheduler
  if Role of node is GROUP_MANAGER_ROLE **then**
    Choose a subset M of member nodes to provide T_i
    Run Task Scheduler if the manager node $\in M$
    Send TREP to members in M
  Remove T_i from $DecisionQueue$
  Process next task in $DecisionQueue$

be considered as a factor in choosing M.

5.4.4 Summary

We present a resource allocation protocol based on the multilevel management schema of the DISON platform. Sensor nodes ask for the support of their manager node to decide to offer an application task. It can reduce the data duplication in dense networks where there are many nodes that have the same sensing capability in a functional area. It also allows a sensor node which does not have enough of the required sensing capabilities to cooperate with other nodes to offer the requested task. This is efficient when the nodes that have more energy can share the work with nodes with lower values. Therefore, the network lifetime can be improved.

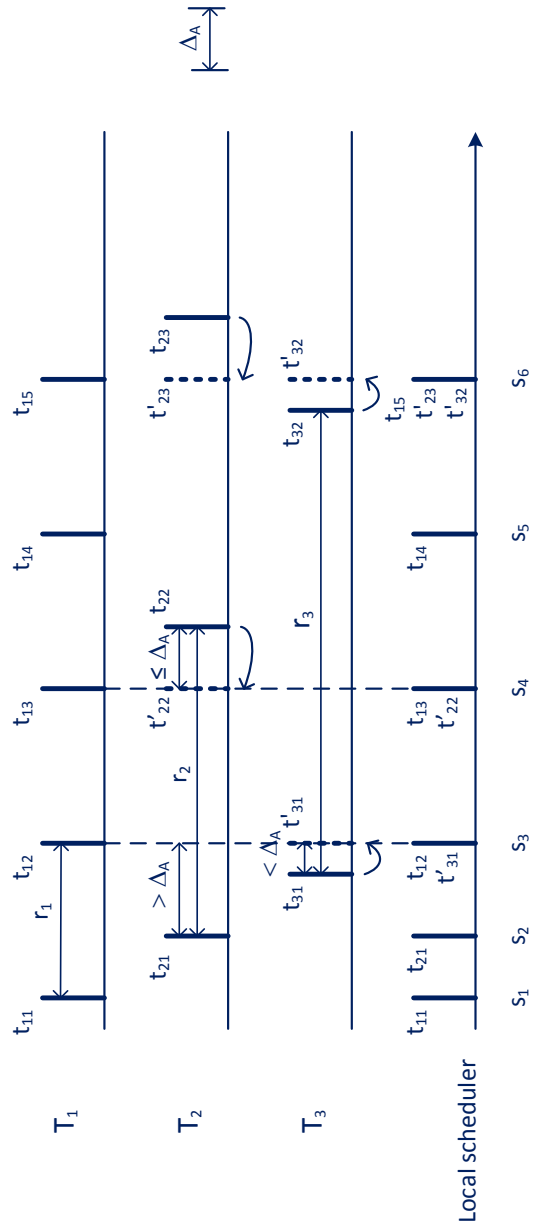


Figure 5.3: Local Task Scheduler

5.5 Local Task Scheduler

We propose a Local Task Scheduler to align all application task executions on sensor nodes in time to accumulate idle time duration in the processor schedule. This scheduler also enables to aggregate sensing data from multiple tasks in order to reduce energy consumption because sending bigger packets consumes less energy than sending multiple smaller packets. The basic idea of the Local Task Scheduler is merging instances of different tasks that are close on the time axis.

5.5.1 Problem formulation

For the ease of presentation, we first describe an example of three simultaneous application tasks as shown in Figure 5.3. These active tasks arrive at different times. The tasks T_1, T_2, T_3 have different sampling rate r_1, r_2, r_3 respectively. A instance t_{ij} is the time that task T_i needs to execute the j^{th} times. For example, T_1 has instances t_{11}, t_{12}, t_{13}, t_{14}, t_{15}.

Task T_1 arrives earliest. It starts its first instance immediately at time s_1 and sets the next run time to $s_1 + r_1$. At time s_2, task T_2 arrives. Since the arriving time of T_2 is far to the next run time of the ongoing tasks, T_2 also starts t_{21} immediately at s_2 and sets the run time for its next instance. The time that task T_3 arrives is close to the time executing t_{12}. Therefore, instead of starting t_{31} immediately, T_3 delays and moves its instances to t'_{31} and t'_{32}. At time s_3, the processor merges the requirements of both tasks T_1 and T_3 (e.g. sensing types) to trigger required hardware to perform the combined task. In the schedule, s_4 is the time to execute only t_{13}. However, since the next run time of t_{22} is very close to s_4, T_2 moves its instances to earlier times t'_{22} and t'_{23}. As a result, there are 2 task instances to execute t_{13} and t'_{22} at s_4. There is no special at time s_5. Only the instance t_{14} is started. After rescheduling at previous times, all of three tasks execute their next instance $(t_{15}, t'_{23}, t'_{32})$ at the same time at s_6.

We now define the problem in general. Given a set of n active tasks $\mathcal{A} = \{T_i\}, i = 1, 2, ..., n$. Each task T_i has n_i instances $\{t_{ij}\}, j =$

$1, 2, ..., n_i$. The interval between two continuous instances of a task T_i is the sampling rate r_i. The time slot to run a instance is s_k, where $k = 1, 2, ..., m, 1 \leq m \leq \sum_{i=1}^{n} n_i$.

Each time slot s_k is corresponding to at least one task instance t_{ij}. The problem is to minimize m, so that the accuracy of every task is still satisfied. To make the problem more clear, we make the following definitions.

Definition 5. Let Δ_A be the acceptable error period. A new task instance t'_{ij} can replace the current instance t_{ij} without affecting the accuracy of the task if $\left|t_{ij} - t'_{ij}\right| \leq \Delta_A$. For example, t'_{31} and t_{31}.

Definition 6. Two instances of two different tasks are *close* if $|t_{ij} - t_{i'j'}| \leq \Delta_A$.

Now, our problem is to find all of the possible *close* task instances, merge them and update the next instances of the corresponding tasks. There are two merging types. The *right merging* happens when there is a new arriving task at a time that is close to the next instance of the existing tasks. For example, t_{31} and t_{12}. The left merging happens when there exists an instance of another task close to the current task instance. For example, t_{13} and t_{22}.

5.5.2 Algorithm design

Algorithm 3 shows the pseudo code of the Local Task Scheduler. In summary, the progress of the Local Task Scheduler includes three step:

- Step 1. Update the next run interval of existing active tasks.
- Step 2. Perform left or right merging if needed.
- Step 3. Choose current task instances to execute.

Algorithm 3 Local Task Scheduler algorithm

When there is a new active task tid, do $UpdateActiveTasks()$
if $nextRunInterval + lastRunTime - currentTime <= \Delta_A$ **then**
    Add task to the list of active tasks $\mathcal{A}$
else
    Add task to $RunningQueue$
if $currentTime == nextRunInterval + lastRunTime$ **then**
    $newRunInterval \leftarrow -1$
    for $i \leftarrow 1, numActiveTasks$ **do**
        $\mathcal{A}[i].nextRunInterval \leftarrow \mathcal{A}[i].nextRunInterval - nextRunInterval$
        if $\mathcal{A}[i].nextRunInterval \leq \Delta_I$ **then**
            Add $\mathcal{A}[i]$ to $RunningQueue$
            $\mathcal{A}[i].nextRunInterval \leftarrow \mathcal{A}[i].samplingRate$
            if $newRunInterval == -1$ **or** $newRunInterval > \mathcal{A}[i].samplingRate$ **then**
                $newRunInterval \leftarrow \mathcal{A}[i].samplingRate$
        else
            if $newRunInterval == -1$ **or** $newRunInterval > \mathcal{A}[i].nextRunInterval$ **then**
                $newRunInterval \leftarrow \mathcal{A}[i].nextRunInterval$
    $nextRunInterval \leftarrow newRunInterval$
    Remove completed tasks

5.5.3 Summary

We formulated a general solution to find and merge instances of different tasks whose start time are close on the time axis. The proposed scheduler allows to merge tasks instances both when a new task arrives and when they are running by using the left merging and the right merging.

Chapter 6

CASE STUDY

This chapter demonstrates the feasibility of the proposed framework and management mechanisms by means of two design examples. The scenarios we have considered for the demonstration are data collection applications in a single sensor network and a shared sensor network. Since these scenarios deal with different application requirements, we also demonstrate the suitability of our management platform to support applications in different domains.

6.1 Data Collection in a Single Sensor Network

The main job of WSNs is collecting data. Therefore, the first scenario we considered is a typical data collection application. Each sensor node samples at a particular frequency and the sampled data is transmitted to the sink through multihops.

6.1.1 Scenario

Figure 6.1 illustrates the basic scenario of the data collection application in a single sensor network. A sink broadcasts queries to sensor nodes in the network. The sensor nodes collect data and transmit back to the sink.

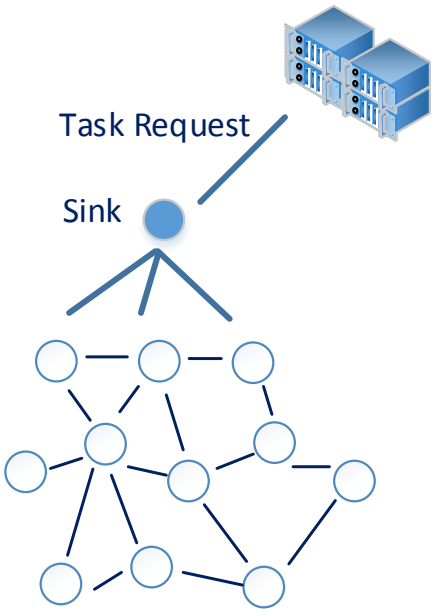


Figure 6.1: Data collection in a single sensor network

6.1.2 Performance evaluation in SENSE

For initial evaluation, we have implemented the above scenario in SENSE [19]. We divide the network area into a grid of multiple adjacent cells. The grid is symmetric or asymmetric. Nodes are deployed randomly in each cell of the grid. We assume that each node can know the identifier of the cell where it is located and its sensing radius covers that cell. For comparison, we have implemented a data collection application where all sensor nodes transmit data to the sink as soon as receiving the query as the baseline, and refer to it as BDC. In BDC, the data relay path is built on the query broadcasting process. Each node setups the node from which it receives the query as the next node to forward data. We also have built the forwarding mechanism in DISON based on the query request similarly to BDC. However, DISON stores the identifier of the cell from which it receives the query instead of the node identifier because DISON could switch off some nodes which can be on the data relay route to save energy. When the node need to send or forward data to the sink, it adds the cell identifier to the data packet and broadcast to its neighbors. If the neighbor nodes are in the indicated cell, they forward the packet. Otherwise, they drop the packet.

In our experiments, we consider one symmetric scenario and one asymmetric scenario. In the symmetric scenario, the network is deployed in a 10x10 grid, in which each cell has a radius of 40m. In the asymmetric one, we use a 10x2 grid, that could be similar to the room structure of an office building floor. Sensor nodes are randomly placed in each cell of the grid with a density from 2 to 4 nodes. Moreover, in order to ensure the reality of the performance results we have also placed sensor nodes randomly through all the area and vary the number of nodes in case of the asymmetric grid. The transmitting radius of a sensor node is 50m. The battery level of each node is generated randomly in the range of $[0, 10^6]$J. We define one special node as the sink and put it randomly in the grid. The sink node is responsible for broadcasting a query at a specific time to collect data from the network. The time to start querying is set randomly in the range $[80, 90]$ seconds. We set the sampling frequency to 10 seconds.

Table 6.1: Network Settings

Parameters	**Value**
Tx Power	$24,75 \cdot 10^{-3}$**J**
Rx Power	$13,5 \cdot 10^{-3}$**J**
Channel	Error-free
Tx Rate	125 kbps
MAC layer	IEEE 802.11 (DCF)
Transmitting Radius	50 m
Simulation Time	3000s
Threshold	0.8

The period of collecting sensing data is chosen randomly in the range $[1000, 1200]$ seconds. Every random number in the simulator is generated by using a linear congruential algorithm and 48-bit integer arithmetic. The simulation results are calculated based on the results from running each scenario with 10 different seed numbers. The other parameters are set as in Table 6.1.

In order to calculate the coverage ratio, we calculate the number of packets including sensing data from each cell at the sink node. We denote recv_i as the number of received packets including the sensing data from i^{th} cell. Assume that each sensor node transmits sensing data to the base station in the interval T seconds with a rate APP_RATE. To make it clearly, T is the query period and APP_RATE is the sensing data sampling frequency. If the condition shown in the Equation 6.1 is satisfied, the i^{th} cell is considered as transmitting enough data to the sink. In other words, i^{th} cell is covered. The simulation area is fully covered if every cell in the simulation area is covered.

$$\text{recv}_i = \text{Threshold} \cdot \left\lceil \frac{T}{\text{APP\_RATE}} \right\rceil \tag{6.1}$$

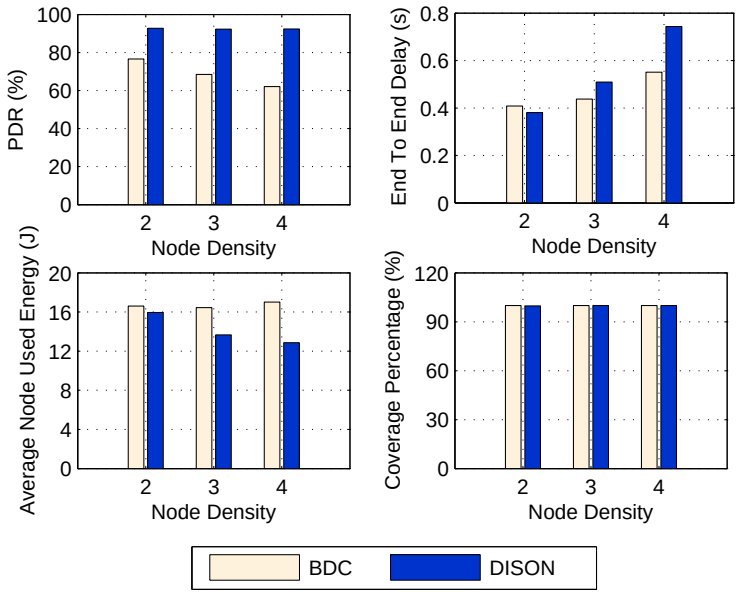


Figure 6.2: Symmetric networks

We use the following metrics to evaluate our proposal:

- **Average Node Used Energy.** The average used energy of each node.
- **Packet Delivery Rate (PDR).** The percentage of packets generated at the sensor nodes that are successfully delivered to the sink.
- **Coverage Percentage.** The percentage of the number of covered cells per total cells in the grid.
- **End-to-End Delay.** The average time taken for a packet to be transmitted across the network from the node to the sink.

In case of the 10x10 grid scenario (Figure 6.2), the packet delivery rate is improved significantly for all node density values, up to 30% at node density 4. The end-to-end delay of the DISON solution is higher

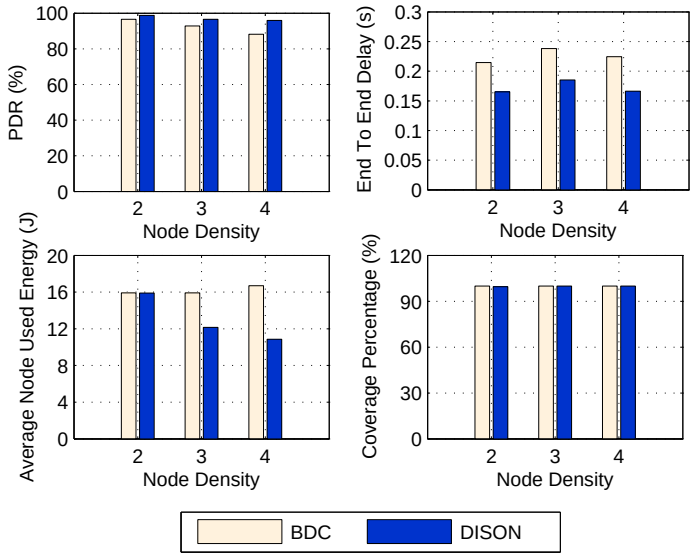
than BDC because there are more successfull received packets in DISON, and these packets can be sent from nodes which are far from the sink. Therefore it results in an increase of the average delay. It is important that the used energy is reduced significantly, up to 23% at node density 4.

As shown in Figure 6.3a, DISON solution does not improve the used energy comparing to BDC if there are only two nodes in each cell. However it increases the packet delivery rate slightly and reduces the end-to-end delay significantly. It is because the energy that saves from switching off some nodes is approximately the same as the required management overhead in DISON. In addition, as the number of nodes that transmit data to the sink decreases, it results in less traffic in the network. Therefore the packet delivery rate and the end-to-end delay are improved. When the node density increases to 3 and 4, we can see that DISON reduces the power consumption from 18% to 29% because there are more redundant nodes that can be switched off without compromising the network operation. Moreover, it also improves significantly the packet delivery rate and the end-to-end delay. In case nodes are randomply placed in all the area, DISON still achieve better performance than BDC (Figure 6.3b). The used energy and the end to end delay of DISON is much lower than BDC while the packet delivery rate is also slightly higher. In addition, it is easy to see that the sensing coverage is ensured in all scenarios.

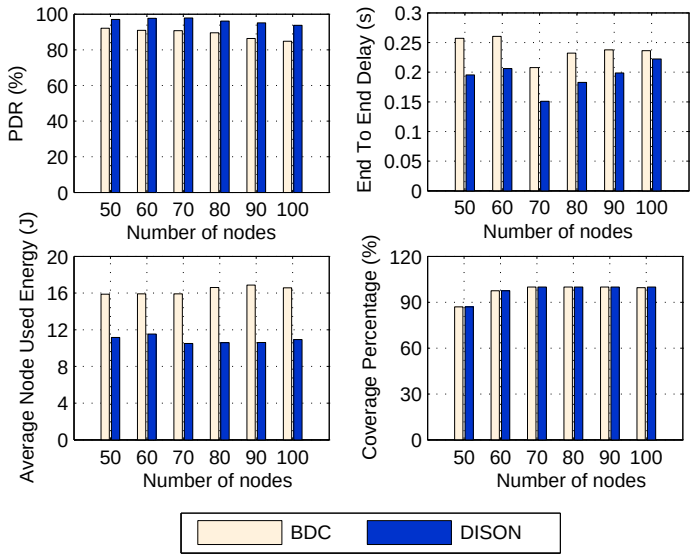
6.1.3 Performance evaluation on Testbed

In order to evaluate the feasibility and the efficiency of the proposed framework, we have implemented DISON in TinyOS 2.1.2 [21], a popular operating system for WSNs. The implementation includes three parts: DISON modules, main application, and user client.

The DISON modules implement the DISON Management and Hardware APIs components in the proposed protocol stack. The connection between DISON modules and network protocols is established in the main application. This design allows the user to configure the DISON framework easily. For example, the user can decide which clustering algorithm to select manager nodes or which routing protocol is used to



(a)



(b)

Figure 6.3: Asymmetric networks: (a) Variable Node density, (b) Variable Number of Nodes

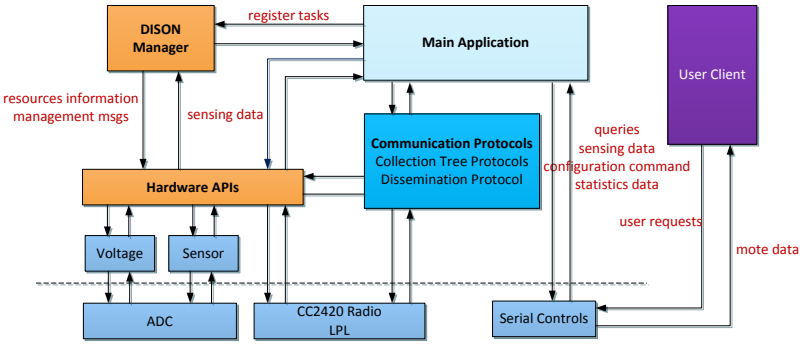


Figure 6.4: DISON implemented blocks

send statistic data to the network manager. The main application makes a demo of using DISON framework to allocate network resources in a simple data collecting application. The user client provides a graphical interface which allows users to set up queries and network configuration requests. It displays the data it receives from motes in both text and chart format. Fig. 6.4 shows our implementation. An example of our graphical interface is shown in Fig. 6.5.

We have used Collection Tree Protocol [88] and Dissemination Protocol [89] which are included in TinyOS 2.x to broadcast or transmit message over a multi hops network. The Collection Tree Protocol is used to transmit data (sensor readings and management statistic data) from sensor nodes to the base station. The Dissemination Protocol is used to broadcast queries and management requests (e.g., reset network and adjust transmitting power) from the user to all the nodes in the network.

In DISON modules, we have defined public interfaces that allows other components to interact with management functions. These interfaces also enable developers to add or update functions easily without changing other components. Each interface includes the declaration of management functions and events. We have defined four public interfaces. The first interface *DISONManagementAppI* includes functions and events which are used to interact with the application components.

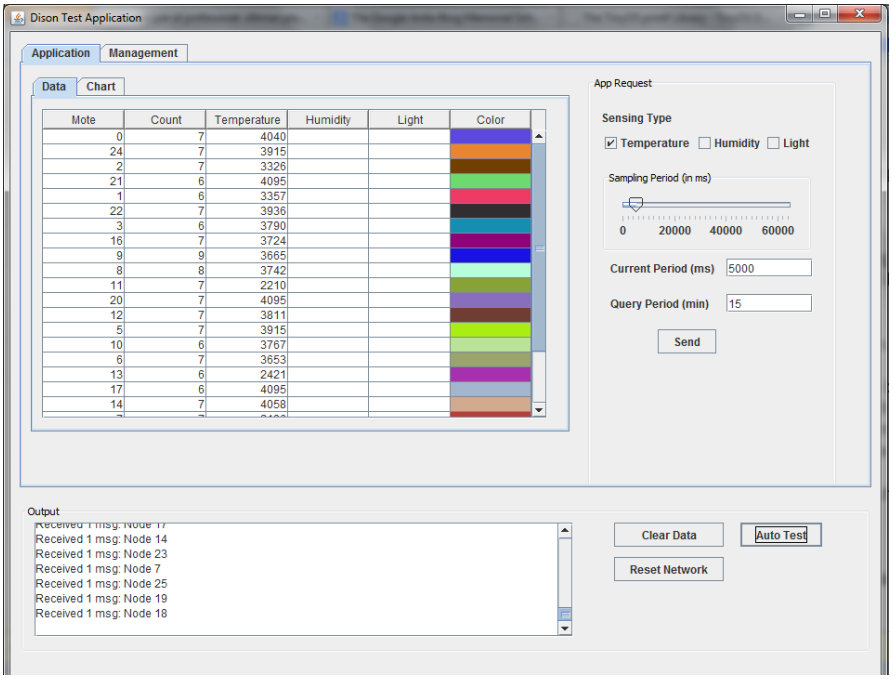


Figure 6.5: Graphical Interface of the user client application

```
interface DISONManagementAppI{
 command error_t registerTask(uint8_t taskid,
    uint8_t sensingtype,
        uint16_t period);
 event void registerDone(uint8_t taskid, bool response);
 command void configure(uint8_t cmd, uint16_t value);
}
```

The second interface *DISONManagementHardwareI* refers to management functions and events related to the node's hardware. It helps management process to be independent of hardware or low level components such as the radio transceiver or the leds. Therefore, the developers only need to update the functions which implement these interfaces to support different node's platforms without affecting to other parts of DISON framework. It enables DISON to support multiple platform easily.

```
interface DISONManagementHardwareI{
 command error_t readResource();
 event void readResourceDone(error_t result, uint16_t data);
 command error_t broadcastBeacon(message_t* msg, uint8_t len);
 command error_t unicastBeacon(am_addr_t dst,
                message_t* msg, uint8_t len);
 event message_t* recieveBeacon(message_t* msg,
                void* payload, uint8_t len);
 command error_t setNodeSleep(uint16_t period);
 command error_t setLocalWakeUpInterval(uint32_t interval);
 command int8_t getRSSI(message_t* msg);
}
```

The third interface *DISONManagementComI* is responsible for controlling network protocols. In current implementation, we only used this interface to store the packet logs. Network protocols (e.g., CTP) can use this interface in their components to record sent and received packets.

```
interface DISONManagementComI{
 command void initLogs();
 command error_t logPacket(uint8_t type, uint32_t value);
 command error_t logBytes(uint8_t type, uint8_t size);
}
```

The management requests from the user are handled in the last interface *DISONManagementUserI*. For example, the user can request to collect statistic data from the whole network by calling the function *collectLog*.

```
interface DISONManagementUserI{
 command error_t collectLog();
}
```

To use the management functions provided by DISON, the developers of WSN applications simply have to add the corresponding interfaces to their components and wire these interfaces to the DISON components. For example, if they want to use the resource allocating function, they add **use interface DISONManagementAppI** and then call the command **registerTask** as illustrated in Figure 6.6.

We have also designed the required management messages which are used in the resource allocating function. Fig. 6.7 shows the structure of these messages [1]. The first byte of the basic management message, namely the Type field, indicates the type of the management message (e.g., Task Register or Task Reply). Because each specific management message has different structure with a different size, the length of management message is variable. We use the next field of the management message, Payload length, to encode the length of the payload.

In the main application, we design the format of the query packet as following:

```
typedef nx_struct {
nx_uint16_t queryID;
nx_uint8_t sensingType;
nx_uint16_t samplePeriod;
nx_uint16_t queryPeriod;
} query_t;
```

[1]The structures of HELLO, ROC, GL, ROD messages are not included because they only include the identifiers of source node and destination node which are already contained in TinyOS packet header

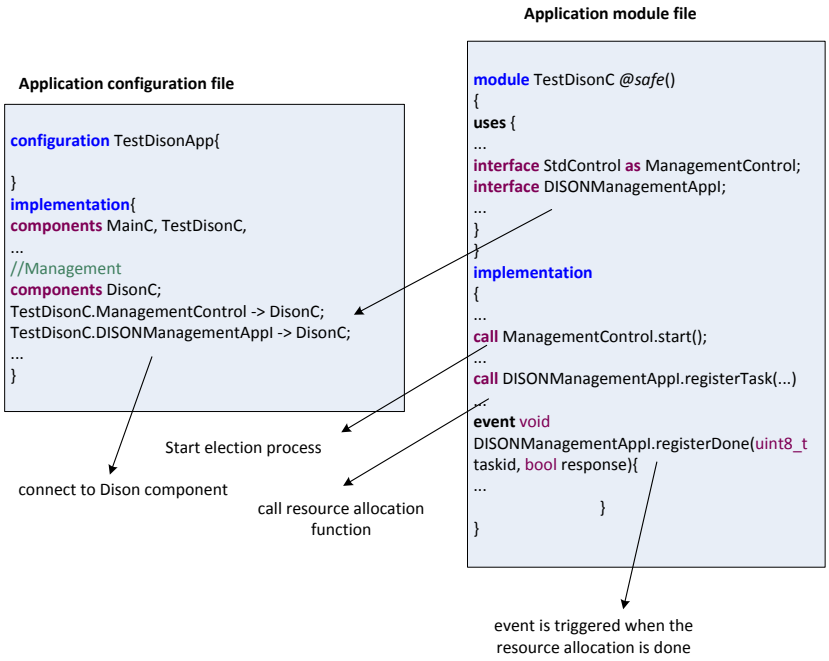


Figure 6.6: An example usage of DISON

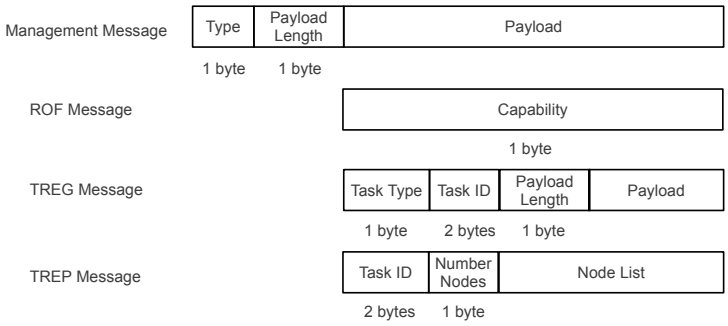


Figure 6.7: Message structure of DISON management messages

The field **sensingType** is used to indicate what sensors are needed to collect data. Each bit of this 8-bit field corresponds to one kind of sensors as shown in Table 4.3. Therefore, when the node receives the query, it simply performs the AND operator between its sensing capability and the value of the field sensingType to know if it has the required sensor. The **samplePeriod** field is the data transmitting rate to the base station. The running period of the query is indicated in the **queryPeriod** field. If the value of this field is 0, it means sensor nodes have to send data to the base station periodically until their batteries are gone or they receive new requests from the base station. The field **queryID** is the identifier of the query.

Deployment Environment

In our evaluation, we have used 26 TelosB motes which are distributed in an apartment of 90 square meters, as shown in Fig. 6.8. Each TelosB mote is equipped with a CC2420 radio and use the default CSMA/CA MAC layer in TinyOS 2.x. There is one mote which is connected to a laptop by an USB cable. This mote roles as the gateway.

We have developed a Java client application to send the application and management requests to the network. In addition, this client application receives and shows the data from the mote on a graphical interface as shown in Fig. 6.5. The users can choose which information they need to collect and send that request to the sensor network. Because our TelosB motes are not equipped with real sensors (e.g., temperature, humidity sensors), we use the DemoSensor component in TinyOS to generate sensing data. The parameters used in our evaluation are summarized in the Table 6.2.

To evaluate the benefits of DISON, we have considered two user applications. In the first user application, all the sensor nodes start to transmit sensing data as soon as they receive the query. In the second application, the resource allocating function supported by DISON framework is used to choose a subset of active nodes. Only these active nodes will collect and transmit sensing data to the sink. Both of applications use CTP as the

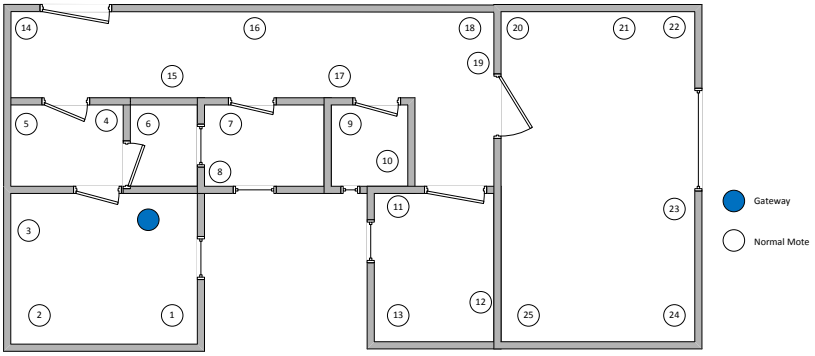


Figure 6.8: Network deployment in an apartment

Parameters	**Value**
TinyOS version	2.1.1
Nodes	26
Platform	TelosB
Radio	CC2420
Battery	2 x Energizer Extreme Rechargeable AA 2300mAh
Java	JDK 1.6.0_38
Maximum Payload	60 bytes
Routing Protocol	Collection Tree Protocol (CTP)

Table 6.2: Evaluation Environment

routing protocol. We call the first application as CTP, and the second one is DISON+CTP.

Performance metrics are used in our evaluation is presented in the below section.

Evaluation Metrics

- **Packet Delivery Ratio (PDR).** The number of distinct received packets divided by the number of packets sent.
- **Duplicate Packets.** The number of duplicate packets that are received at the sink. We added a sequence number to each data packet, and counted the number of received packets which have the same sequence number.
- **Control Overhead.** The number of exchanged management packets and routing packets.
- **Management Overhead.** The number of exchanged management packets.
- **Network Traffic.** The total number of packets sent and received in the network.
- **Energy Consumption.** Because the voltage readings decrease when the battery discharge, they reflect the battery's level of charge. We have estimated node's energy consumption by monitoring the decrease in the voltage reading. TelosBs have a voltage sensor which can be read from an 12-bit ADC (Analog Digital Converter) interface. As defined in [90], the ADC value is converted to a voltage reading with the following formula:

$$V = \frac{\text{ADC Value}}{4096} \cdot V_{ref} \cdot 2 \tag{6.2}$$

in which V_{ref} is the maximum voltage of the batteries. For example, if we use 2xAA 1.5V batteries, the value of V_{ref} is 3 V. All nodes

Module	Memory Size
Clustering algorithm	5.5 KB
DISON Management Modules	36.1 KB
Main Application + DISON	36.762 KB in ROM, 5.706 KB in RAM

Table 6.3: Module code size

in the testbed are fully recharged before each experiment. Since the transformation of the voltage reading is not stable in short time (i.e., it can go up and down depending on the working load or environment conditions), we ran the experiments which evaluate the energy consumption for a long time (15 hours) to ensure that a decrease in the voltage reading is caused by the battery consumption and not by the measurement fluctuation. In these experiments, sensor nodes transmit the voltage reading to the sink. The energy consumption is the difference between the first and the last voltage readings in the experiment.

DISON Code Size

As discussed above, the management system for WSNs must be light weight due to the limited resources of sensor nodes. Table 6.3 shows the code size of DISON implementation in TinyOS. The size of DISON modules is small, approximately 36.1 kB. In addition, when DISON is integrated into the user application, the memory usage is still small.

Network Performance

In order to evaluate the network performance of DISON, we broadcast a query with a sampling period of 5 seconds to collect data during 15 minutes. The transmitting power of the node is set to its maximum value, 0 dBm. In this scenario, all nodes can communicate directly with each other, which results in a high collisions probability. Therefore, the performance of CTP is affected. As shown in Fig. 6.9a, CTP only achieves 90%

PDR. Moreover, the number of duplicate packets[2] is also high. However, we can see that using DISON these problems of CTP can be overcome. The PDR of DISON+CTP is higher than CTP, almost equal to 100%. The number of duplicate packets in DISON+CTP is reduced significantly. The reason why using DISON improves the performance of CTP is that DISON switches off some nodes, which reduces the amount of traffic transmitted over the network and, in turn, the number of collisions. It results in an increase of the number of received packets at the sink, compared to the number of transmitted packets, and a decrease of number of duplicate packets.

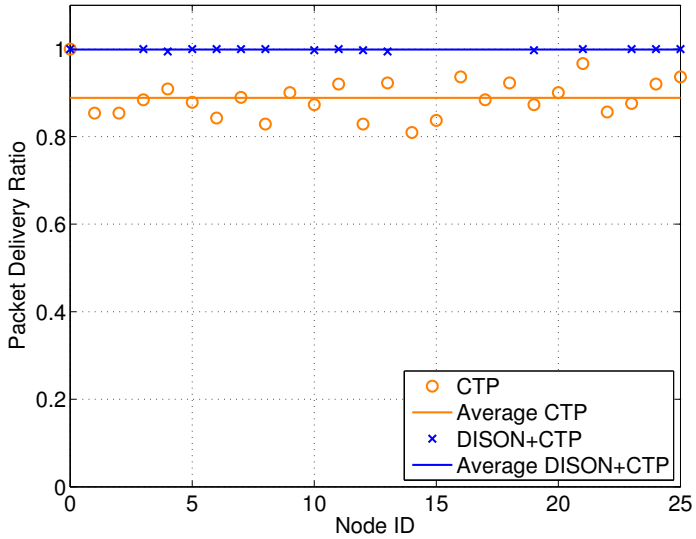
Fig 6.10a shows the communication overhead of each node. The overhead of DISON+CTP is higher than CTP because of the management messages. However, the total traffic of DISON is lower than CTP (see Fig. 6.10b and Fig. 6.10c). It means that using DISON does not represent a higher traffic load.

Impact of transmitting power

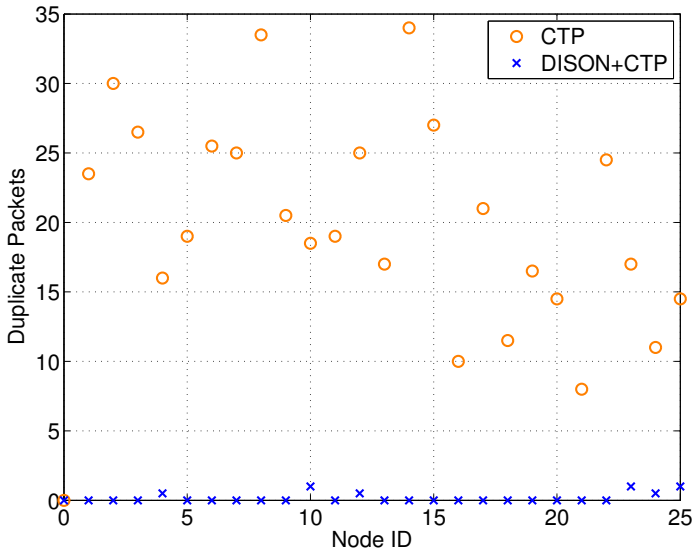
Now we analyze the performance of DISON framework with different transmitting powers. As mentioned in [91], the CC2420 has 8 discrete power levels 31, 27, 23, 19, 15, 11, 7, 3 corresponding to output power values 0, -1, -3, -5, -7, -10, -15, -25 dBm, respectively. We ran our experiments with three power levels 31, 15 and 3. Fig. 6.11 shows the network topology when the transmitting power is -25 dBm and -7 dBm. In case of the maximum transmitting power, 0 dBm, all the nodes can communicate with each other. In each experiment, the sink broadcasts 20 queries. Each query is sent every 40 minutes to collect sensing data every 5 seconds during 30 minutes. The total time of each experiment is approximately 15 hours.

As mentioned in [92], the packet delivery rate of CTP depends on the network topology. The impact of network topology on the network per-

[2]One of disadvantages of CTP is that when the ACK is lost due to the network collision, especially in high density networks, the sender will resend the frame to the receiver. It results in the packet duplication.

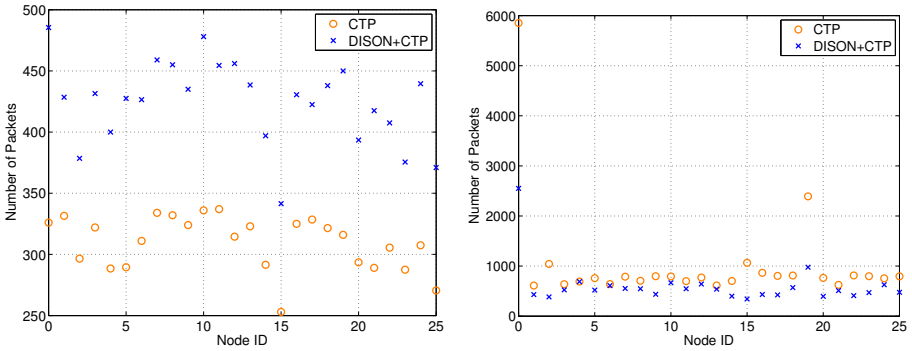


(a)



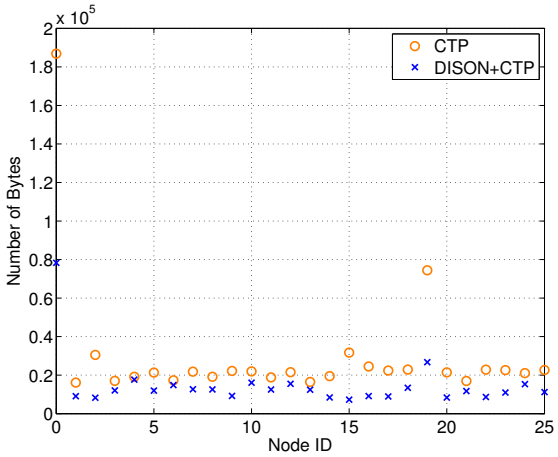
(b)

Figure 6.9: (a) Packet Delivery Ratio, (b) Duplicate Packets



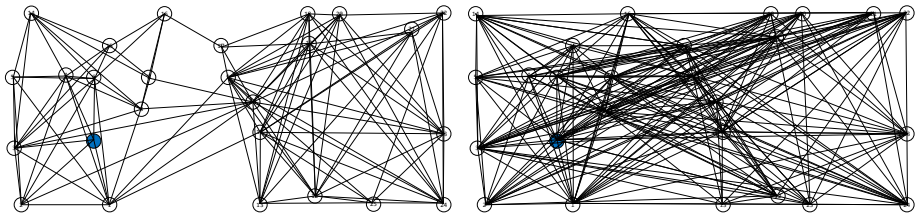
(a)

(b)



(c)

Figure 6.10: (a) Control Overhead, (b) Network traffic in number of packets, (c) Network traffic in number of bytes



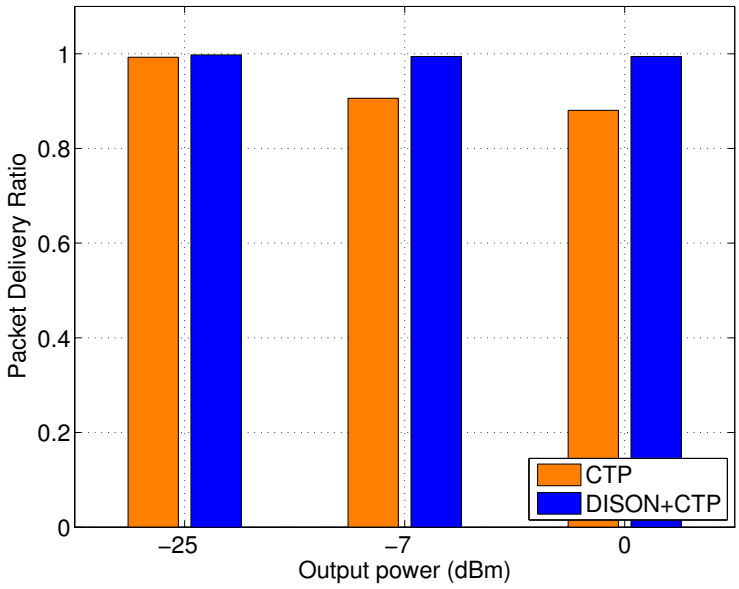
(a) (b)

Figure 6.11: (a) Transmitting Power -25 dBm, (b) Transmitting Power -7 dBm

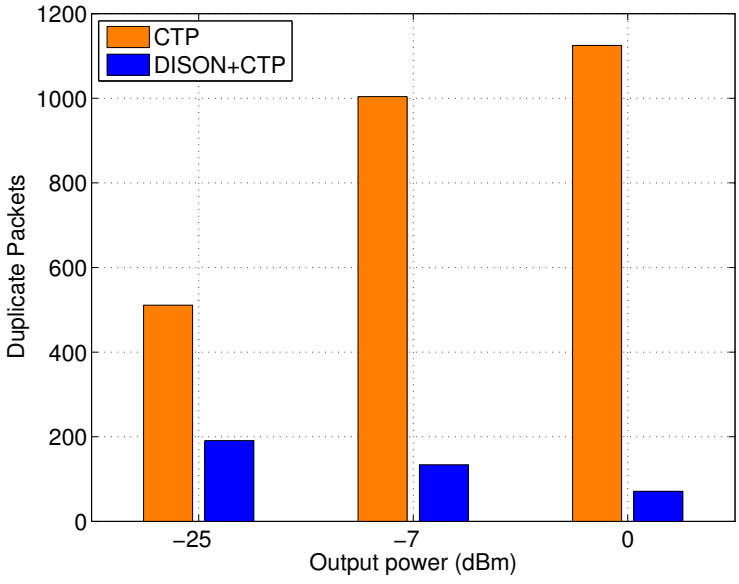
formance is also shown in our experiments. We can see the variability of the PDR when the network topology changes. However, using DISON can improve the PDR of CTP in case it does not have good performance. In Fig. 6.12a, the PDR of CTP without and with using DISON is almost similar because when the transmitting power is small, the number of neighbor nodes of each sensor node is small (i.e., see Fig. 6.11a), which results in a low collision probability. Therefore, the PDR in both cases is almost 100%. However, when the transmitting power increases, the network traffic in the same collision domain increases, which causes more collisions and packet loss. The result is that the PDR of the original CTP is reduced. When using DISON, some nodes are switched off. It helps to reduce the traffic. Therefore, it improves the PDR of the original CTP.

Although DISON does not improve the PDR in case of a low transmitting power, it still helps to reduce the number of duplicate packets since it uses less network traffic. Especially, DISON reduces significantly the number of duplicate packets in case of high density networks.

Next, we analyze the energy consumption of the nodes to show the benefits of our proposed framework. Fig. 6.13a shows that using DISON reduces the energy consumption in all cases. In Fig. 6.13b, 6.13c and 6.13d, it is easy to see that the energy consumption of most of nodes in case of using DISON is smaller than the original CTP. However, there are some nodes in DISON+CTP consuming more energy than in CTP. The reason is that when some nodes are switched off, the network topology



(a)



(b)

Figure 6.12: (a) Packet Delivery Rate, (b) Duplicate Packets

changes and those nodes have to forward more packets. Therefore, they consume more energy.

Impact of the manager election cycle

In this section, we consider the impact of the manager election cycle to choose the manager nodes dynamically. In these experiments, the transmitting power is set to -7 dBm. The sink broadcasts 20 queries with parameters similar to the used ones in the previous experiments. We ran the manager node election process every 1, 2, 4 queries. We also ran the same experiment in which DISON is not used, that is, the election round is not done. Fig. 6.14 shows that using DISON always improves the PDR. Moreover, the energy consumption is reduced significantly when the election cycle is large (see Fig. 6.15a). The reason is that the number of management packets is reduced when the manager election process does not run frequently. It is shown clearly in Fig. 6.15b. However, since manager nodes have to perform management tasks, they consume more energy than normal nodes. Therefore, it would be necessary to elect new manager nodes after a period of working time to ensure the balance of network resources.

Scalability analysis

In order to evaluate the scalability of DISON framework, we have deployed a new testbed in the second floor of the Tanger building in the Poble Nou campus at Universitat Pompeu Fabra, in Barcelona. We run experiments with a different number of TelosB motes, namely 20, 40, 60 and 80 motes placed in a corridor with offices at each side as illustrated in Figure 6.16. The density of nodes per square meter was kept constant for the case of 20 and 40 nodes. However, for 60 and 80 nodes, since they were deployed in the same area used for the case with 40 nodes, the network density was higher.

In these experiments, the latest version of TinyOS is used. The sink node is connected to a computer placed in one of the offices through an USB cable. In each experiment, the sink broadcasts a query to collect

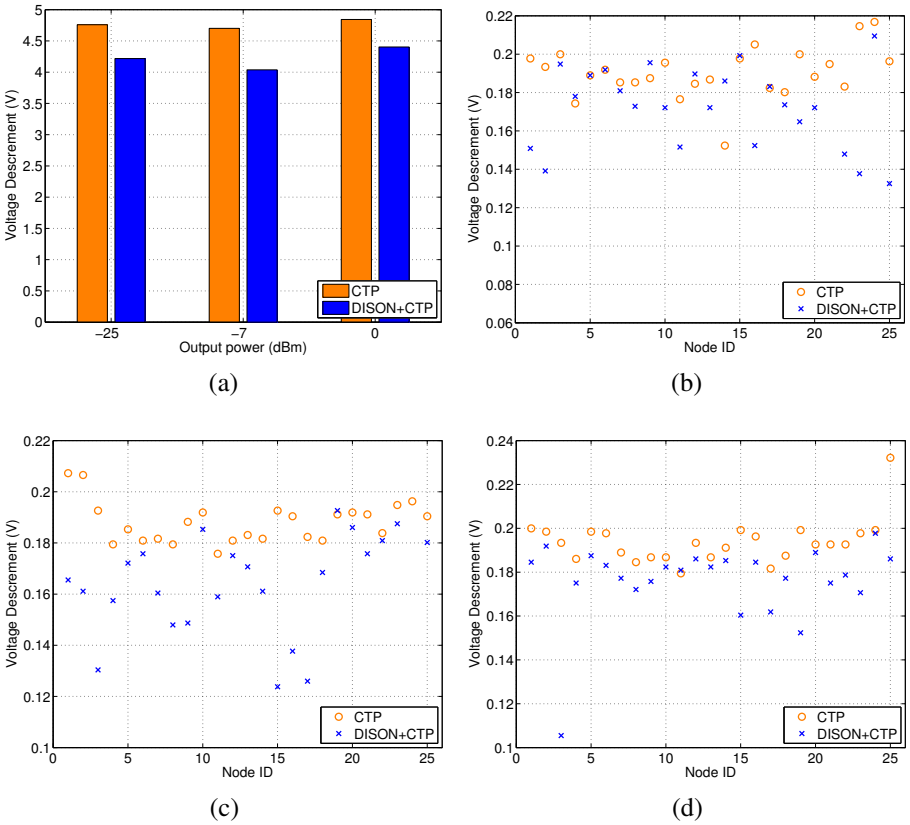


Figure 6.13: (a) Energy Consumption vs Radio Range, (b) Energy Consumption vs Node ID when transmitting at -25 dBm, (c) Energy Consumption vs Node ID when transmitting at -7 dBm, (d) Energy Consumption vs Node ID when transmitting at 0 dBm

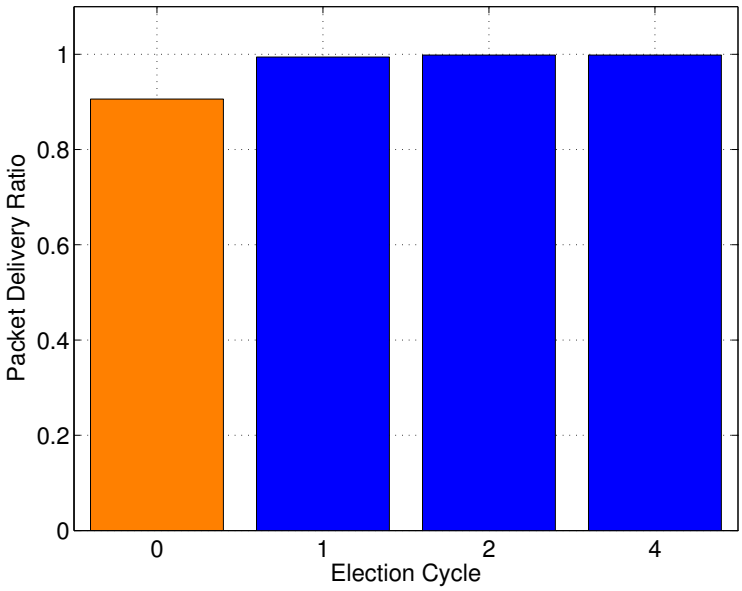
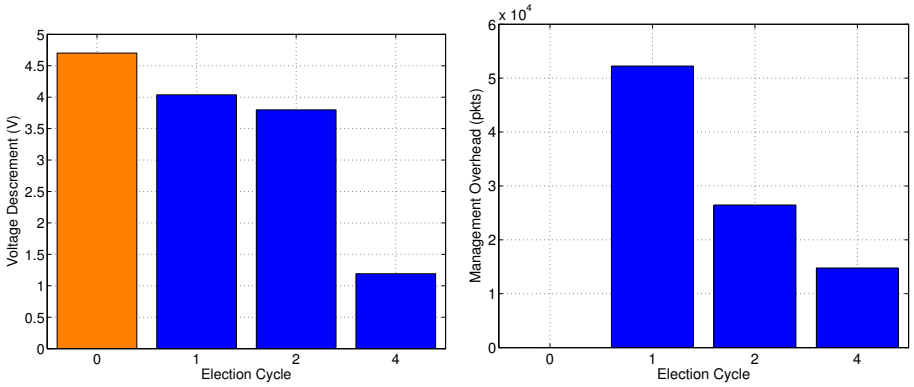


Figure 6.14: Packet Delivery Rate vs Election Cycle



(a) (b)

Figure 6.15: (a) Energy Consumption vs Election Cycle, (b) Management Overhead vs Election Cycle

data every 5 seconds during 15 minutes. The DISON framework is used to choose a subset of sensor nodes that will remain active and will be responsible for collecting and transmitting data to the sink. In these experiments, the transmitting power is set to -25 dBm. In addition, each experiment is repeated 3 times to achieve reliable results.

Figure 6.16: Testbed deployment on a building floor

Number of nodes	**Number of clusters**	**AVG number of nodes per cluster**	**Number of management packets**	**AVG number of management packets per cluster**
20	5	4	985.67	197.13
40	10	4	2040.67	204.07
60	13.33	4.5	4591.33	344.35
80	12	6.67	7116.67	593.06

Table 6.4: Scalability analysis results. The number of management packets is the sum of all transmitted, received and overhear packets

Table 6.4 shows the management overhead of DISON framework for different number of sensor nodes. As expected, the management overhead increases with the number of nodes, as there are more nodes performing management processes. However, if we observe the average number of management packets per cluster, we can see that it only depends on the cluster size (i.e., the number of nodes that belong to the same cluster), which in turn is related to the network density. From those results, we can conclude that DISON is scalable, as regardless the number of nodes in the network, it is able to kept the management overhead per cluster constant if the network density is not altered. In that situation, metrics such as the energy consumption or the network lifetime would not be severely affected.

6.2 Data Collection in a Shared Sensor Network

Suppose we would like to extend the scenario presented in section 6.1 to support a shared sensor network where there are multiple applications and multiple sinks.

6.2.1 Scenario

We discussed in Section 1.2 the potential of the shared sensor network infrastructure which supports multiple applications belong to different authorities. The general scenario of a shared sensor network infrastructure is illustrated in Figure 6.17. There are several sinks deployed around a sensor network. Users can use PC, smart phone or tablet to send task requests directly to sensor networks at different location. The problem is how sensor nodes handle these requests efficiently. In this thesis, we use management functions in DISON framework to allocate network resources and schedule the run time for multiple application tasks.

When a new task arrives, the sensor node execute the resource allocation protocol to find out if it needs to support this task. The resource

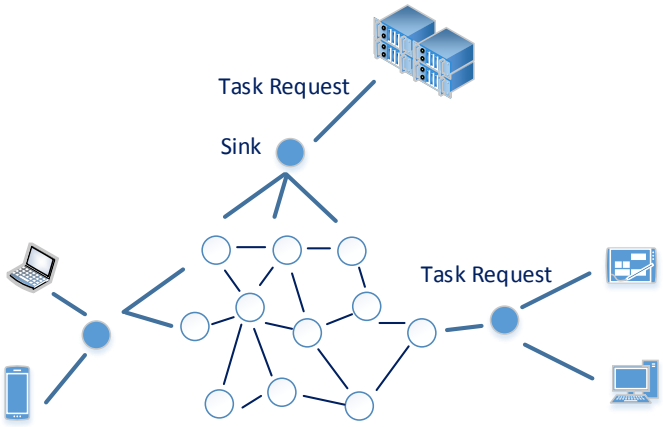


Figure 6.17: Data collection application in a share sensor network

allocation protocol decides which nodes host the requested task according nodes' resources. After the decision process, the Local Task Scheduler refreshes the current schedule of running tasks to cover the new one. During the run time, the Local Task Scheduler combines *close* task instances. This allows to reduce the computing processes by combining similar requests. For example, there are 2 queries. One requests to collect temperature and light each $5s$. Another requests to collect temperature and humidity each $20s$. The Local Task Scheduler combines 2 queries each 20. Therefore, sensor node only needs to call temperature sensor one time. Since shared sensor networks can have a lot of traffic to support different applications, the ACK mechanism is used to increase the reliability of the management messages exchange.

6.2.2 TinyOS Interface

We have extended the implementation of the DISON framework which presented in the previous case study to support the multiple queries scenario. We added the following functions to the interface *DISONManage-*

mentAppI support the new proposed mechanisms. The function *getCurrentTasks* is used to get all tasks that need to start collecting sensing data while the function *getActiveTasks* returns the list of tasks that needs to transmit data to the sink. The function *triggerSensors* allows to trigger a set of sensors to collect data from the environment. When the collecting process has done, the event *collectDone* is signaled.

```
interface DISONManagementAppI{
 ...
 command error_t getCurrentTasks(uint8_t* sensingtype,
        uint16_t* nextrunperiod);
 command void getActiveTasks(nx_uint16_t* tasklist);
 command void triggerSensors(uint8_t sensingtype);
 event void collectDone(uint16_t* val);
}
```

Since application tasks can come from different sinks, the global identification of each task is represented as the pair of the sink identification sid and the local task identification lid at that sink, that is, (sid, lid). To save the memory and reduce the communication overhead, we use the Canto pairing function [93] to generate an unique number from a pair of two integer numbers. Therefore, the message structure of the query task is the same as presented in the first case study, where the $queryID$ field is the output of the Canto pairing function with the inputs sid and lid.

6.2.3 Performance evaluation in TOSSIM

We firstly evaluate the effectiveness of the proposed mechanisms through simulations in TOSSIM, a simulation tool in TinyOS. We have compared the network performance of six applications. In the first application **MQ**, only the local schedule mechanism is used. The second **DA2** and the third **DA3** applications are similar to the first one, however, they aggregate the data from two and three concurrent tasks respectively before transmitting to the sink. Three last applications combine both the local schedule mechanism and the resource allocation mechanism in handling multiple concurrent tasks. We name them as **RS**, **RSDA2**, and **RSDA3**. In all ap-

Parameter	Value
Δ_{ACK}	1s
Δ_A	1s
MAX_GROUP_MEMBERS	10
MAX_ACTIVE_TASKS	5
MAX_RETRIES_TIMES	5

Table 6.5: Simulation Parameters

plications, five queries are broadcast sequentially each 20 seconds. Their periods are 1900s, 1800s, 1700s, 1600s, 1500s respectively. Therefore, there is a maximum of 5 concurrent task on a sensor node. The maximum packet size is set to 40 bytes. Other parameters are summarized in the Table 6.5.

All applications use CTP as the routing protocol. Ten network topology are generated for each network size and each protocol are executed ten times with each network topology. Each result is the average of 100 experiments. The network performance is evaluated based on the following performance metrics:

- **Packet Delivery Ratio (PDR).** The number of distinct received packets divided by the number of packets sent.
- **Duplicate Packets.** The number of duplicate packets that are received at the sink. We added a sequence number to each data packet, and counted the number of received packets which have the same sequence number.
- **Management Overhead.** The number of exchanged management packets.
- **Network Traffic.** The total number of packets exchanged in the network.
- **Response Time.** The average time sinks receive the first packets of each application from all nodes for all tasks.

Impact of the sampling rate

In order to evaluate the impact of the sampling rate, two set of sampling rates are used. The first set includes sampling rates that have common divisor $\{5, 10, 15, 20, 25\}$. In the second set, sampling rates are generated randomly in the range of $[5, 30]$. We generate randomly networks of 65 nodes in an area of 50x50m2. The communication range of each sensor node is set to 15m. The sink is placed randomly in the area. It broadcasts 5 queries with the sampling rate from two above sets correspondingly.

In case of the common divisor sampling rates, the probability of occurring simultaneous tasks is high. If the data from simultaneous tasks is not aggregated in one packet, it can result in high traffic in a time slot. Thus, the packet can be lost due to the collision. Using the RS mechanism and data aggregation can overcome this problem since they reduce the number of data packets. This can be seen in the Figure 6.18a. The PDR of **MQ** in case of the common divisor sampling rates is lowest. It is only approximately 93%) while the one in case of the random sampling rates is $\simeq 98\%$. The PDR of **RSDA3** in both sampling rates are higher than 99%, though. The Figure 6.18a also shows that our RS mechanism can improve the PDR regardless of sampling rates. Moreover, our RS mechanism reduces the network traffic and the duplicate packets significantly. Especially, in case of the common divisor sampling rates, using RS and data aggregation can reduce more than 60% of the network traffic and more than 50% of the duplicate packets (Figure 6.18c and Figure 6.18b). However, the response time when using RS is higher than without RS (Figure 6.18d). This is because the network needs time to allocate resources. The network traffic of **RSDA3** is little higher than **RSDA2** in case of the random sampling rates. The reason is the PDR of **RSDA3** is higher than **RSDA2** while the occurring probability of three simultaneous queries is low.

Impact of network density

Next we evaluate the impact of node density on the efficiency of our proposed mechanisms. In the simulations, the network area is set to

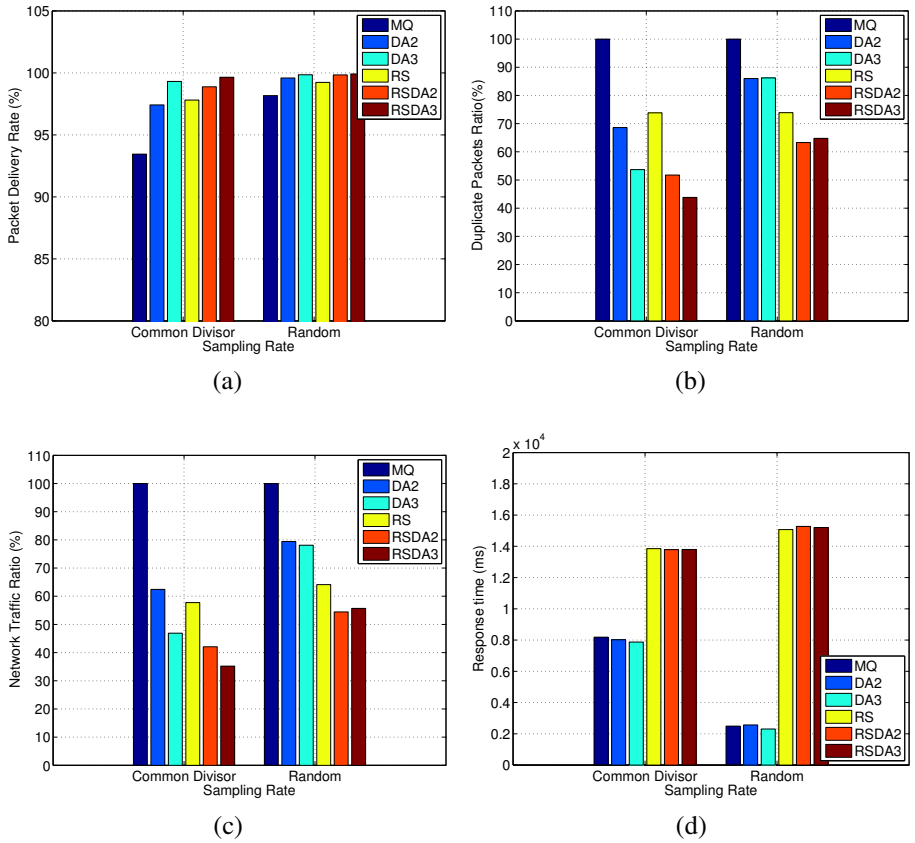


Figure 6.18: (a) Packet Delivery Ratio (b) Duplicate Packets (c) Communication Overhead (d) Response Time

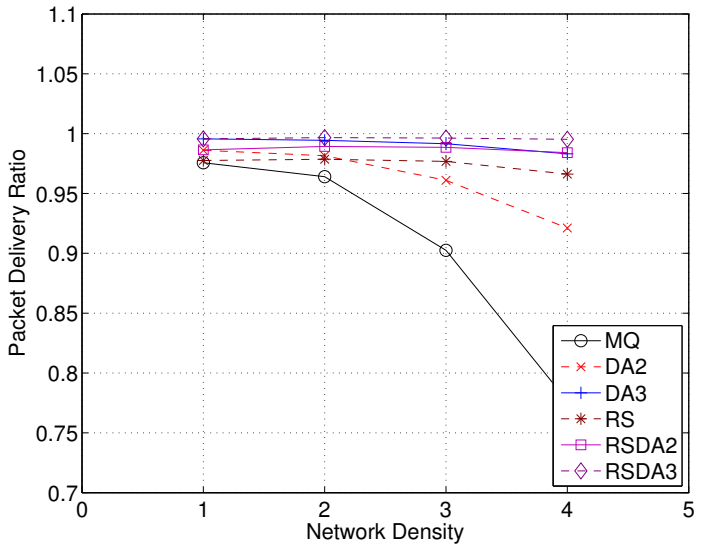
50x50m2. The area is divided into 25 cells. The number of nodes in each cell is equal and varies from 1 to 4. The communication range of each node is set to 15m. The sink broadcasts 5 queries with sampling rates from the set $\{5, 10, 15, 20, 25\}$.

When the node density increases, the data traffic increases. It results in a higher probability of collision. As a result, the PDR of MQ decreases significantly. Using RS and data aggregation mechanism can overcome this problem. As seen in Figure 6.19a, the PDR is improved. Especially, the PDR of RSDA3 is always approximately 100%. Although the management overhead increases, the network traffic of applications using RS is still much lower than the ones without using RS (Figure 6.20a). The network traffic of RSDA3 is only around $1/4$ of MQ. The number of duplicated packets is also reduced significantly, 50% in case of RSDA3 (Figure 6.19b). Figure 6.20b shows that the response time when using RS is higher, however, the time difference decreases when the network density increases.

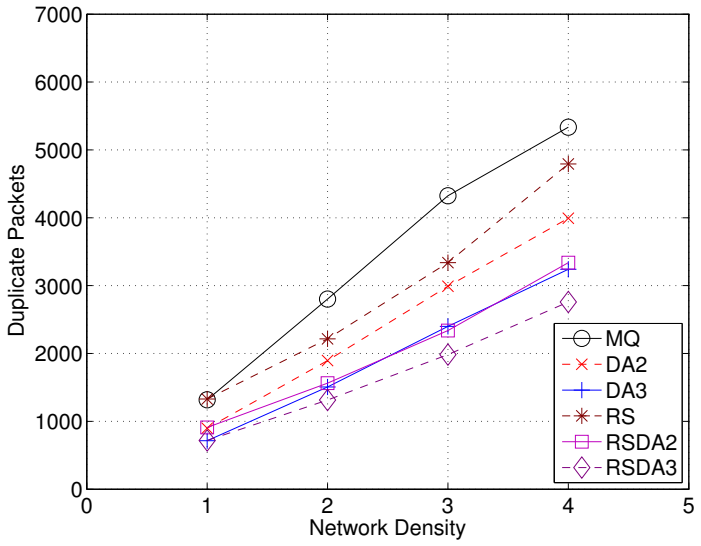
Impact of network scale

In this section, we evaluate the impact of network scale. The network size varies from 50x50m2 to 50x250m2. The network area is divided into cells. The size of each cell is 25x25m2. There are three nodes in each cell. Therefore, the number of nodes are 12, 24, 36, 48 and 60 nodes respectively. Each node has a communication range of 20m. There is only one sink which are placed randomly in the network area. The sink broadcasts 5 queries continuously with sampling rates from the set $\{5, 10, 15, 20, 25\}$.

As seen in Figure 6.22a, the network scale does not affect the PDR. However, the number of duplicate packets increases when the network scale is bigger. Figure 6.23a shows that using RS can improve the network performance in large scale networks. Figure 6.22 and Figure 6.23 indicate that applications using RS have higher PDR, less duplicated packets and less total network traffic. The management overhead in case of network scale is similar to the case of network density (Figure 6.24). However,

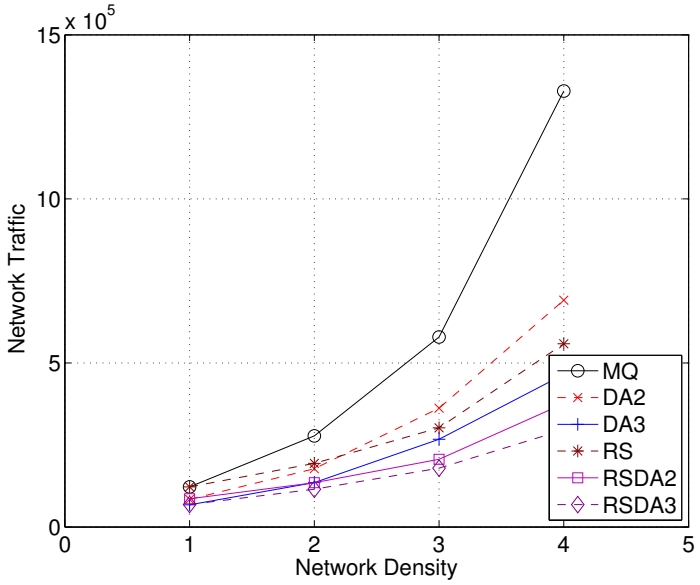


(a)

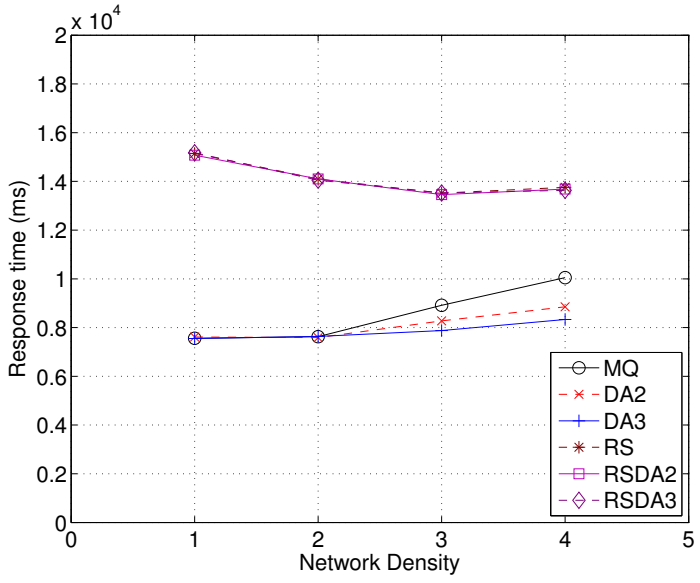


(b)

Figure 6.19: (a) Packet Delivery Ratio (b) Duplicate Packets



(a)



(b)

Figure 6.20: (a) Communication Overhead (b) Response Time

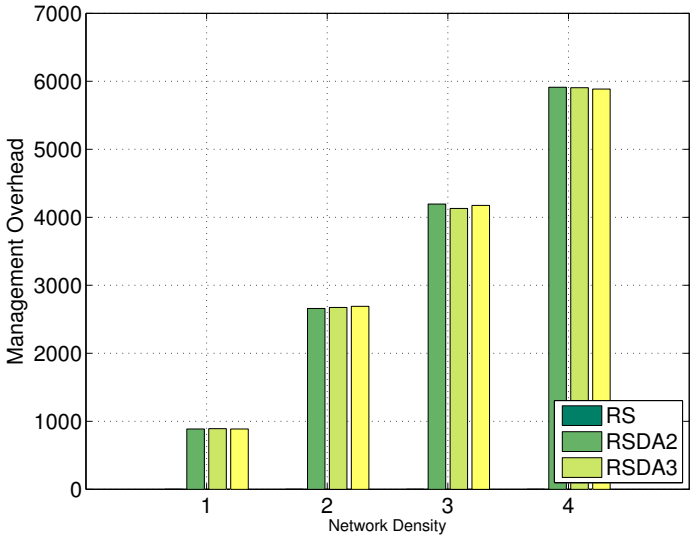
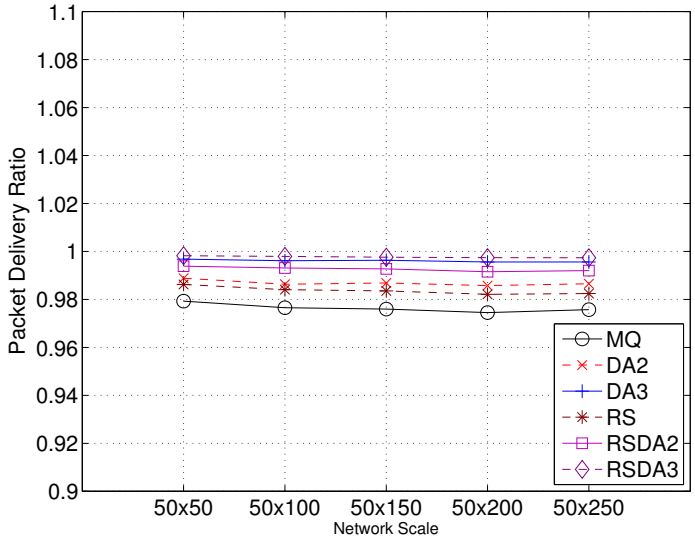


Figure 6.21: Management Overhead

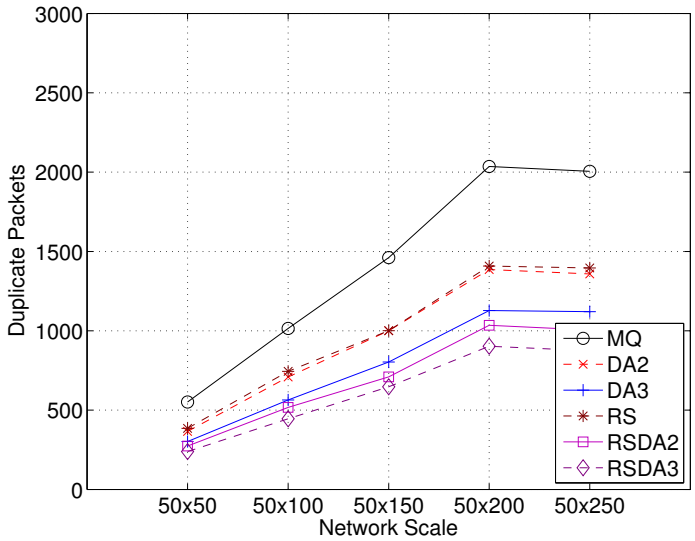
since total network traffic when using RS is smaller than without using RS, the increase of management overhead is acceptable. In the Figure 6.23b, the response time fluctuates when the network size changes. In case of MQ and RS, the response time increases when the network size increases because it takes more time to transmit packets from the nodes which are far from the sink. However, it goes up and down in other cases since the data aggregation can reduce the response time in some situations.

Impact of multiple sinks

The next evaluations is to evaluate the network performance when there are multiple sinks. In the simulations, 200 sensor nodes are deployed in an area of 200x200m2. The network is divided into 100 cells. Each cell has two nodes. The position of each node is randomly. The communication range is set to 40m. The number of sinks varies from 1 to 3. Each sink is placed randomly in the network area. In case of one sink, the sink

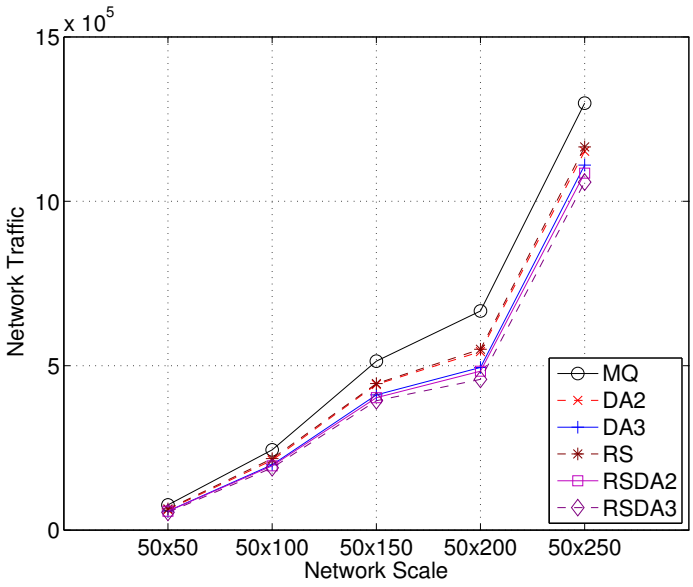


(a)

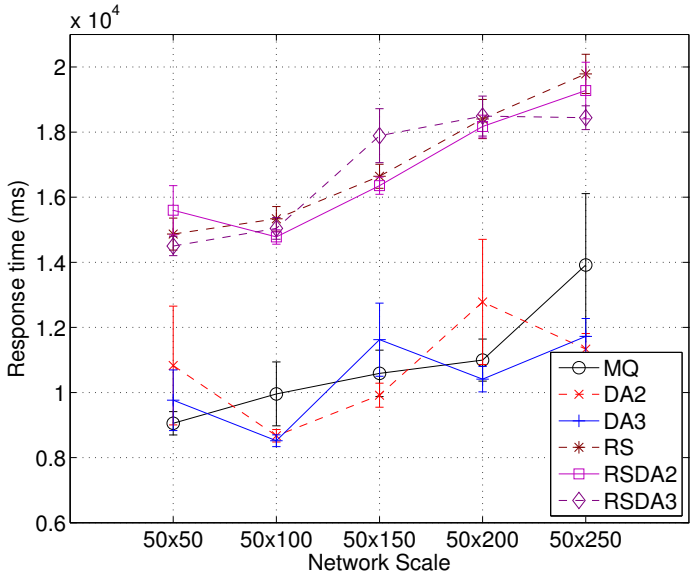


(b)

Figure 6.22: (a) Packet Delivery Ratio (b) Duplicate Packets



(a)



(b)

Figure 6.23: (a) Communication Overhead (b) Response Time

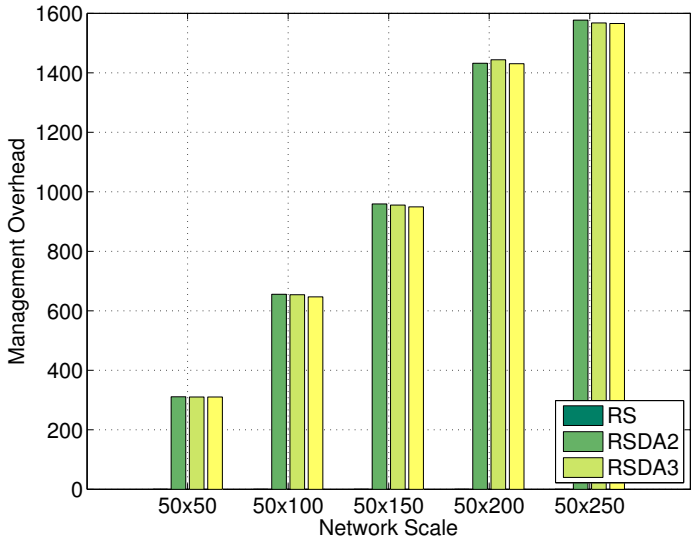
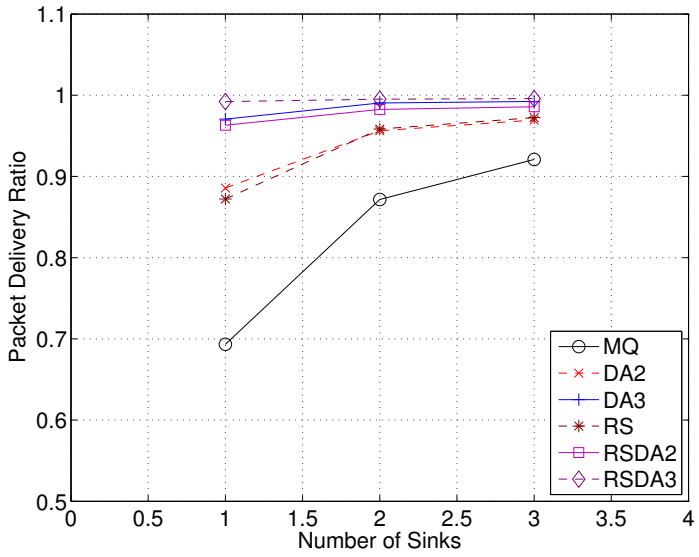


Figure 6.24: Management Overhead

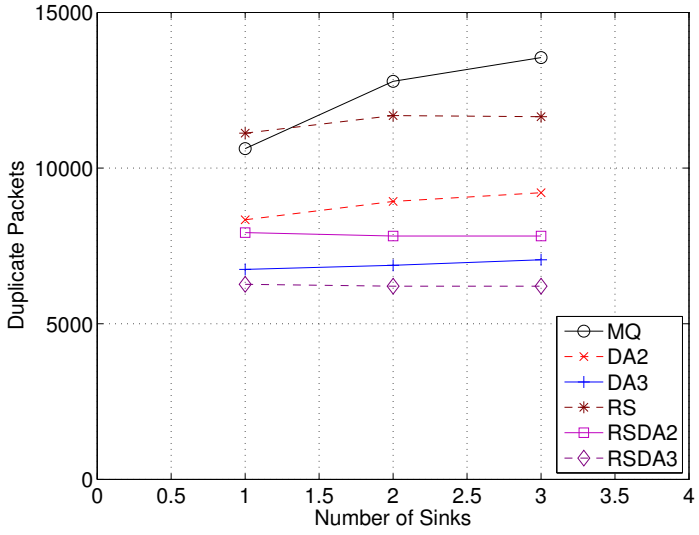
broadcasts all 5 queries with sampling rates 5, 10, 15, 20, 25 respectively. In case of two sinks, the first sink broadcasts the first three queries while the second sink is responsible for the last two ones. In case of three sinks, two first queries is sent by the first sink, the next two ones comes from the second sink, and the last one is sent by the third sink.

In CTP protocol, the data packets are forwarded to the nearest sink. We assume that sinks can communicate directly through a high speed communication media such as WIFI or 4G. If the data packet is not for that sink, the sink can forward it to the correct destination easily.

Figure 6.25a, 6.26a, and 6.26b show that using multiple sinks can help to increase the PDR, reduce network traffic and response time. The reason is multiple sinks are placed around the network while the routing protocol (CTP) forwards the data packets to the nearest sink. This helps to reduce the number of hops needed to reach a sink. However, the number of duplicated packets increases when the number of sinks increases because of the higher PDR (Figure 6.25b).

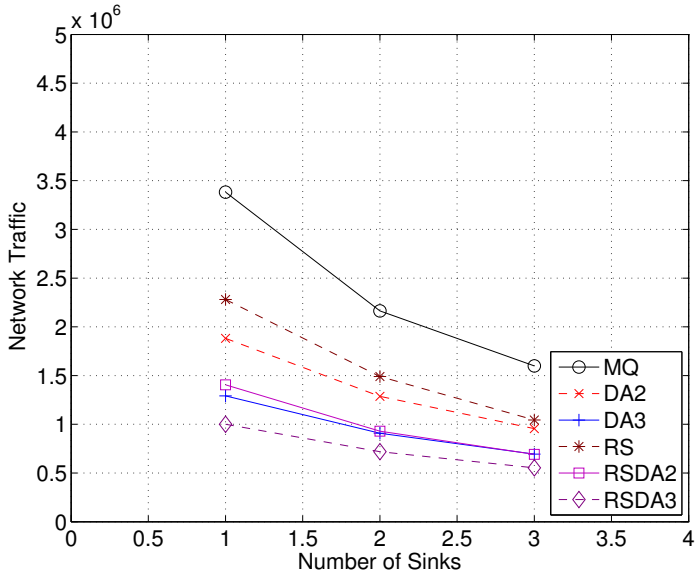


(a)

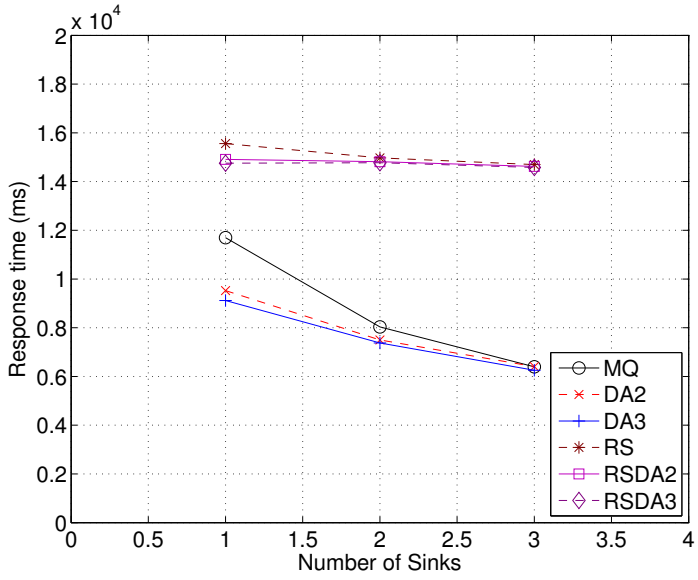


(b)

Figure 6.25: (a) Packet Delivery Ratio (b) Duplicate Packets



(a)



(b)

Figure 6.26: (a) Communication Overhead (b) Response Time

Chapter 7

CONCLUSIONS AND FUTURE WORKS

In this chapter we present the conclusions of the work presented in this thesis, and we identify the topics that we recommend for future work.

7.1 Conclusions

This thesis has presented a self organizing management platform for WSNs that is capable to meet the increasing needs of WSN applications. By utilizing a multilevel management schema and an independent layer, the platform is scalable and flexible. We have developed a role election protocol to assign management roles to sensor nodes. This protocol allows the developer or the network administrator to change the manager candidate requirements in both the development and run time stages.

We have defined the Context model and the Policy model to store specific information and events that can cause a management function. These two models are designed simple to be suitable for a limited resource device as sensor node. We have also developed a context detection and policy based reasoning mechanism to combine with the data models to support the adaptability.

To validate the efficiency of the platform, we have designed a resource

allocation protocol which coordinate network resources among adjacent sensor nodes to support multiple application tasks from different authorities. A local task scheduler has also been developed to efficiently share local node resources across multiple independent application tasks. Additionally, we have defined a general Task model that cover both requested and running tasks in local node or in the member nodes that the node manages.

Since not all of sensor node platform support the Acknowledgment mechanism at MAC layer, we have developed a high level ACK mechanism to reduce the packet loss in dense networks. This mechanism is independent to management functions and communication protocols. Therefore, it is easy to enable or disable this ACK mechanism.

We have demonstrated the suitability of our proposed platform by means of case studies. In the first case study, we considered a typical data collection in a single WSN. We have customized the resource allocation protocol to choose a set of active nodes to support the query task while scheduling sleeping time for others. We have designed management components in SENSE and in nesC, a programming language of TinyOS. These components are independent. Each component provides and uses public interfaces to/from other ones. Evaluation processes are perform on both simulator and testbed with a large number of experiments. The results show that the proposed platform not only reduces the node's energy consumption but also improves other network performance metrics such as the packet delivery rate and the end-to-end delay.

In the second case study, we validate the feasibility and the efficiency of the platform when there are simultaneous applications from the same sink or different sinks in a shared WSN. In this scenario, the resource allocation protocol is customized to choose a set of nodes to support a requested application task. The local task scheduler is enable to identify when a task instance starts. We have also developed a simple data aggregation mechanism that combines data from multiple tasks before sending to the sink. We have extended the implementation in the first case study to support new management functions. We have varied both network topology and network parameters to see the impact of the proposed platform.

Our experiment results showed that the proposed platform reduces significantly the network traffic, hence, leading to consuming less energy.

7.2 Future Works

The work started by this dissertation can be further continued in many directions. They are summarized as below:

- The number of WSN applications is increasing rapidly. A lot of applications require to sense events and report them to the sink in an efficient and timely manner. A model that described the Quality of Service for various applications should be defined. Each application should have a priority according to its required QoS. When multiple applications are executed concurrently in a single WSN, the WSN management system should combine the QoS models of all applications and generate a global QoS model to find the general trade-off in case the required QoS of all running applications can not be guaranteed. There should be a mechanism to arrange the execution of application tasks to meet the overall QoS. The existing resource allocation protocol and the local task scheduler mechanism should be improved to cover the QoS issues.

- As mentioned previously, WSNs is one of key components of IoT. The technologies that will enable sensor nodes to Internet directly are being developed and tested. For example, the 6LowPAN standard, defined by IETF [94], enables the use of IPv6 packets in restricted networks such as sensor networks. It is necessary to address management issues when sensor nodes have Internet connectivity.

- One of the most interesting direction to work on in the future is the development of protocols based on DISON framework to manage power in harvesting sensor networks. Using energy harvesting as a complement power source for battery is one of research directions in WSNs that have attracted a lot of researchers recently. However, there are some limitations of the harvesting energy source.

For example, the the harvested energy availability typically varies with time in a nondeterministic manner and different nodes may have different harvesting opportunity [95]. There should be a new data model that stores information from harvesting energy sources. Such a data model may be used to find out the energy transformation cycle (e.g. at what time in the day the energy is harvested more or which location of node can collect more energy). The existing resource allocation protocol should be also improved to be aware the harvesting energy.

Bibliography

[1] A. Caragliu, C. Del Bo, and P. Nijkamp, "Smart Cities in Europe," *Journal of urban technology*, vol. 18, no. 2, pp. 65–82, 2011.

[2] K. Su, J. Li, and H. Fu, "Smart City and The Applications," in *Electronics, Communications and Control (ICECC), 2011 International Conference on*, pp. 1028–1031, Sept 2011.

[3] H. Chourabi, T. Nam, S. Walker, J. Gil-Garcia, S. Mellouli, K. Nahon, T. Pardo, and H. J. Scholl, "Understanding Smart Cities: An Integrative Framework," in *System Science (HICSS), 2012 45th Hawaii International Conference on*, pp. 2289–2297, Jan 2012.

[4] "Open Cities." http://opencities.net/.

[5] "Smart Santander." http://www.smartsantander.eu/.

[6] T. Naumowicz, R. Freeman, H. Kirk, B. Dean, M. Calsyn, A. Liers, A. Braendle, T. Guilford, and J. Schiller, "Wireless Sensor Network for habitat monitoring on Skomer Island," in *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, pp. 882–889, Oct 2010.

[7] M. Bocca, J. Toivola, L. Eriksson, H. J., and H. Koivo, "Structural Health Monitoring in Wireless Sensor Networks by the Embedded Goertzel Algorithm," in *Cyber-Physical Systems (ICCPS), 2011 IEEE/ACM International Conference on*, pp. 206–214, April 2011.

[8] D. Surie, O. Laguionie, and T. Pederson, "Wireless sensor networking of everyday objects in a smart home environment," in *Intelligent Sensors, Sensor Networks and Information Processing, 2008. ISSNIP 2008. International Conference on*, pp. 189–194, Dec 2008.

[9] R. Wang, L. Zhang, R. Sun, J. Gong, and L. Cui, "EasiTia: A Pervasive Traffic Information Acquisition System Based on Wireless Sensor Networks," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 12, no. 2, pp. 615–621, 2011.

[10] M. Bottero, B. D. Chiara, and F. Deflorio, "Wireless Sensor Networks for Traffic Monitoring in A Logistic Centre," *Transportation Research Part C: Emerging Technologies*, vol. 26, no. 0, pp. 99 – 124, 2013.

[11] A. Majeed and T. Zia, "Multi-set Architecture for Multi-applications Running on Wireless Sensor Networks," in *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*, pp. 299–304, 2010.

[12] I. Leontiadis, C. Efstratiou, C. Mascolo, and J. Crowcroft, "SenShare: Transforming Sensor Networks into Multi-Application Sensing Infrastructures," in *Wireless Sensor Networks*, pp. 65–81, Springer, 2012.

[13] C. Cano, B. Bellalta, A. Sfairopoulou, and M. Oliver, "Low Energy Operation in WSNs: A Survey of Preamble Sampling MAC Protocols," *Computer Network*, vol. 55, pp. 3351–3363, Oct. 2011.

[14] P. Santi, "Topology Control in Wireless Ad hoc and Sensor Networks," *ACM Computing Surveys*, vol. 37, pp. 164–194, June 2005.

[15] Y. Manolopoulos, D. Katsaros, and A. Papadimitriou, "Topology Control Algorithms for Wireless Sensor Networks: A Critical Survey," in *Proceedings of the 11th International Conference on Computer Systems and Technologies and Workshop for PhD Students in*

Computing on International Conference on Computer Systems and Technologies, CompSysTech '10, (New York, NY, USA), pp. 1–10, ACM, 2010.

[16] S. K. Singh, M. Singh, and D. Singh, "Routing Potocols in Wireless Sensor Networks–A Survey," *International Journal of Computer science and engineering Survey (IJCSES)*, vol. 1, no. 2, pp. 63–83, 2010.

[17] U. Prathap, P. D. Shenoy, K. Venugopal, and L. Patnaik, "Wireless Sensor Networks Applications and Routing Protocols: Survey and Research Challenges," in *Cloud and Services Computing (ISCOS), 2012 International Symposium on*, pp. 49–56, IEEE, 2012.

[18] P. Horn, "Autonomic computing: IBM's Perspective on the State of Information Technology," 2001.

[19] G. Chen, J. Branch, M. Pflug, L. Zhu, and B. Szymanski., "SENSE: A Sensor Network Simulator," *Advances in Pervasive Computing and Networking*, pp. 249–267, 2004.

[20] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, pp. 126–137, ACM, 2003.

[21] P. Levis, "TinyOS 2.0 Overview." http://www.tinyos.net/ dist-2.0.0/tinyos-2.0.0/doc/html/overview.html.

[22] S. Sudevalayam and P. Kulkarni, "Energy Harvesting Sensor Nodes: Survey and Implications," *Communications Surveys Tutorials, IEEE*, vol. 13, no. 3, pp. 443–461, 2011.

[23] J. Case, "Management of High Speed Networks with The Simple Network Management protocol (SNMP)," in *Local Computer Networks, 1990. Proceedings., 15th Conference on*, pp. 195–199, Sep 1990.

[24] W. C. Nitin, W. Chen, N. Jain, and S. Singh, "ANMP: Ad hoc Network Network Management Protocol," *IEEE Journal on Selected Areas in Communications*, vol. 17, pp. 1506–1531, 1999.

[25] L. B. Ruiz, J. M. Nogueira, and A. A. F. Loureiro, "MANNA: A Management Architecture for Wireless Sensor Networks," *Communications Magazine, IEEE*, vol. 41, no. 2, pp. 116–125, 2003.

[26] M. Younis, M. Youssef, and K. Arisha, "Energy-aware Management for Cluster-based Sensor Networks," *Computer Networks*, pp. 649–668, 2003.

[27] H. Song, D. Kim, K. Lee, and J. Sung, "UPnP-Based Sensor Network Management Architecture," in *The Second International Conference on Mobile Computing and Ubiquitous Networking*, Apr. 2005.

[28] "Universal Plug and Play (UPnP)." http://tools.ietf.org/html/rfc6970.

[29] G. Tolle and D. Culler, "Design of an Application-Cooperative Management System for Wireless Sensor Networks," in *Wireless Sensor Networks, 2005. Proceeedings of the Second European Workshop on*, pp. 121 – 132, jan.-2 feb. 2005.

[30] F. Yuan, W.-Z. Song, N. Peterson, Y. Peng, L. Wang, B. Shirazi, and R. LaHusen, "A Lightweight Sensor Network Management System Design," in *Pervasive Computing and Communications, 2008. PerCom 2008. Sixth Annual IEEE International Conference on*, pp. 288–293, 2008.

[31] W. L. Lee, A. Datta, and R. C. Oliver, "WinMS: Wireless Sensor Network-Management System, An Adaptive Policy-Based Management for Wireless Sensor Networks," tech. rep., School of Computer Science & Software Engineering, The University of Western Australia, 2006.

[32] S.-H. Cha, J.-E. Lee, M. Jo, H. Y. Youn, S. Kang, and K.-H. Cho, "Policy-Based Management for Self-Managing Wireless Sensor Networks," *IEICE Transactions*, pp. 3024–3033, 2007.

[33] T. Le, W. Hu, S. Jha, and P. Corke, "Design and Implementation of a Policy-based Management System for Data Reliability in Wireless Sensor Networks," in *Local Computer Networks, 2008. LCN 2008. 33rd IEEE Conference on*, pp. 762 –769, oct. 2008.

[34] Y. Zhu, S. L. Keoh, M. Sloman, E. Lupu, Y. Zhang, N. Dulay, and N. Pryce, "Finger: An Efficient Policy System for Body Sensor Networks," in *Mobile Ad Hoc and Sensor Systems, 2008. MASS 2008. 5th IEEE International Conference on*, pp. 428–433, Sept 2008.

[35] Y. Zhu, S. L. Keoh, M. Sloman, and E. Lupu, "A Lightweight Policy System for Body Sensor Networks," *Network and Service Management, IEEE Transactions on*, vol. 6, pp. 137 –148, september 2009.

[36] Z. Wenbo and X. Haifeng, "A Policy Based Wireless Sensor Network Management Architecture," in *Intelligent Networks and Intelligent Systems (ICINIS), 2010 3rd International Conference on*, pp. 552 –555, nov. 2010.

[37] N. Matthys and W. Joosen, "Towards Policy-based Management of Sensor Networks," in *Proceedings of the 3rd International Workshop on Middleware for Sensor Networks*, MidSens '08, (New York, NY, USA), pp. 13–18, ACM, 2008.

[38] T. Bourdenas and M. Sloman, "Starfish: Policy Driven Self-management in Wireless Sensor Networks," in *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '10, (New York, NY, USA), pp. 75–83, ACM, 2010.

[39] X. Jiang, J. Taneja, J. Ortiz, A. Tavakoli, P. Dutta, J. Jeong, D. Culler, P. Levis, and S. Shenker, "An Architecture for Energy

Management in Wireless Sensor Networks," *SIGBED Rev.*, vol. 4, pp. 31–36, July 2007.

[40] V. Sharma, U. Mukherji, V. Joseph, and S. Gupta, "Optimal Energy Management Policies for Energy Harvesting Sensor Nodes," *Wireless Communications, IEEE Transactions on*, vol. 9, pp. 1326–1336, April 2010.

[41] T. Bourdenas, K. Tei, S. Honiden, and M. Sloman, "Autonomic Role and Mission Allocation Framework for Wireless Sensor Networks," in *Self-Adaptive and Self-Organizing Systems (SASO), 2011 Fifth IEEE International Conference on*, pp. 61–70, Oct 2011.

[42] J. Waterman, G. W. Challen, and M. Welsh, "Peloton: Coordinated Resource Management for Sensor Networks," in *Proceedings of the 12th Conference on Hot Topics in Operating Systems*, HotOS'09, (Berkeley, CA, USA), pp. 9–9, USENIX Association, 2009.

[43] S. Misra and A. Jain, "Policy Controlled Self-configuration in Unattended Wireless Sensor Networks," *Journal of Network and Computer Applications*, vol. 34, no. 5, pp. 1530 – 1544, 2011. Dependable Multimedia Communications: Systems, Services, and Applications.

[44] A. Erdogan, E. Cayirci, and V. Coskun, "Sectoral Sweepers for Sensor Node Management and Location Estimation in Ad Hoc Sensor Networks," in *Proceedings of the 2003 IEEE Conference on Military Communications - Volume I*, MILCOM'03, (Washington, DC, USA), pp. 555–560, IEEE Computer Society, 2003.

[45] C.-L. Fok, G.-C. Roman, and C. Lu, "Agilla: A Mobile Agent Middleware for Self-adaptive Wireless Sensor Networks," *ACM Trans. Auton. Adapt. Syst.*, vol. 4, pp. 16:1–16:26, July 2009.

[46] R. Lopes, F. Assis, and C. Montez, "MASPOT: A Mobile Agent System for Sun SPOT," in *Autonomous Decentralized Systems*

(ISADS), 2011 10th International Symposium on, pp. 25–31, March 2011.

[47] M. Haghighi and D. Cliff, "Multi-agent Support for Multiple Concurrent Applications and Dynamic Data-Gathering in Wireless Sensor Networks," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2013 Seventh International Conference on*, pp. 320–325, July 2013.

[48] C. Muldoon, G. O'Hare, M. O'Grady, and R. Tynan, "Agent Migration and Communication in WSNs," in *Parallel and Distributed Computing, Applications and Technologies, 2008. PDCAT 2008. Ninth International Conference on*, pp. 425–430, Dec 2008.

[49] E. Arseneau, R. Goldman, A. Poursohi, R. B. Smith, and J. Daniels, "Simplifying The Development of Sensor Applications," *Object-Oriented Programming Systems, Languages and Applications (OOPSLA 2006)*, 2006.

[50] W. Heinzelman, A. Murphy, H. Carvalho, and M. Perillo, "Middleware to Support Sensor Network Applications," *Network, IEEE*, vol. 18, pp. 6–14, Jan 2004.

[51] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TinyDB: An Acquisitional Query Processing System for Sensor Networks," *ACM Trans. Database Syst.*, vol. 30, pp. 122–173, Mar. 2005.

[52] T. Liu and M. Martonosi, "Impala: A Middleware System for Managing Autonomic, Parallel Sensor Systems," in *Proceedings of the Ninth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '03, (New York, NY, USA), pp. 107–118, ACM, 2003.

[53] E. Souto, G. Guimarães, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz, and J. Kelner, "Mires: A Publish/Subscribe

Middleware for Sensor Networks," *Personal Ubiquitous Comput.*, vol. 10, pp. 37–44, Dec. 2005.

[54] C. Seeger, K. Van Laerhoven, J. Sauer, and A. Buchmann, "A Publish/Subscribe Middleware for Body and Ambient Sensor Networks that Mediates between Sensors and Applications," in *Healthcare Informatics (ICHI), 2013 IEEE International Conference on*, pp. 199–208, Sept 2013.

[55] P. Costa, G. Coulson, R. Gold, M. Lad, C. Mascolo, L. Mottola, G. Picco, T. Sivaharan, N. Weerasinghe, and S. Zachariadis, "The RUNES Middleware for Networked Embedded Systems and its Application in a Disaster Management Scenario," in *Pervasive Computing and Communications, 2007. PerCom '07. Fifth Annual IEEE International Conference on*, pp. 69–78, March 2007.

[56] K. Shah and M. Kumar, "DReL: A Middleware for Wireless Sensor Networks Management using Reinforcement Learning Techniques," in *Proceedings of the 5th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks*, MidSens '10, (New York, NY, USA), pp. 1–7, ACM, 2010.

[57] F. Ganz, P. Barnaghi, F. Carrez, and K. Moessner, "Context-aware Management for Sensor Networks," in *Proceedings of the 5th International Conference on Communication System Software and Middleware*, COMSWARE '11, (New York, NY, USA), pp. 6:1–6:6, ACM, 2011.

[58] G. Di Modica, F. Pantano, and O. Tomarchio, "SNPS: An OSGi-Based Middleware for Wireless Sensor Networks," in *Advances in Service-Oriented and Cloud Computing* (C. Canal and M. Villari, eds.), vol. 393 of *Communications in Computer and Information Science*, pp. 1–12, Springer Berlin Heidelberg, 2013.

[59] O. Alliance, "Open Service Gateway Initiative, OSGi (2013)." http://www.osgi.org/.

[60] S. Hachem, A. Pathak, and V. Issarny, "Service-oriented Middleware for Large-scale Mobile Participatory Sensing," *Pervasive Mob. Comput.*, vol. 10, pp. 66–82, Feb. 2014.

[61] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin, "Sympathy for The Sensor Network Debugger," in *Proceedings of the 3rd international conference on Embedded networked sensor systems*, SenSys '05, (New York, NY, USA), pp. 255–267, ACM, 2005.

[62] W. Zhao, Y. Liang, Q. Yu, and Y. Sui, "H-WSNMS: A Web-Based Heterogeneous Wireless Sensor Networks Management System Architecture," in *Proceedings of the 2009 International Conference on Network-Based Information Systems*, NBIS '09, (Washington, DC, USA), pp. 155–162, IEEE Computer Society, 2009.

[63] A. A. Abbasi and M. Younis, "A Survey on Clustering Algorithms for Wireless Sensor Networks ," *Computer Communications*, vol. 30, no. 14 - 15, pp. 2826 – 2841, 2007.

[64] Y. P. Chen, A. L. Liestman, and J. Liu, "Clustering Algorithms for Ad Hoc Wireless Networks," in *Ad Hoc and Sensor Networks. Nova Science Publishers*, 2004.

[65] O. Younis and S. Fahmy, "HEED: A Hybrid, Energy-Efficient, Distributed Clustering Approach for Ad Hoc Sensor Networks," *Mobile Computing, IEEE Transactions on*, vol. 3, pp. 366 – 379, oct.-dec. 2004.

[66] P. Marron, A. Lachenmann, D. Minder, J. Hahner, R. Sauter, and K. Rothermel, "TinyCubus: A Flexible and Adaptive Framework for Sensor Networks," in *Wireless Sensor Networks, 2005. Proceeedings of the Second European Workshop on*, pp. 278 – 289, jan.-2 feb. 2005.

[67] M. Kushwaha, I. Amundson, X. Koutsoukos, S. Neema, and J. Sztipanovits, "OASiS: A Programming Framework for Service-Oriented Sensor Networks," in *Communication Systems Software and Middleware, 2007. COMSWARE 2007. 2nd International Conference on*, pp. 1–8, 2007.

[68] L. Mottola and G. P. Picco, "Programming Wireless Sensor Networks: Fundamental Concepts and State of the Art," *ACM Comput. Surv.*, vol. 43, pp. 19:1–19:51, Apr. 2011.

[69] M. Valero, S. S. Jung, A. S. Uluagac, Y. Li, and R. Beyah, "Di-Sec: A distributed security framework for heterogeneous wireless sensor networks," in *INFOCOM, 2012 Proceedings IEEE*, pp. 585–593, IEEE, 2012.

[70] M. Valero, S. Uluagac, S. Venkatachalam, K. Ramalingam, and R. Beyah, "The Monitoring Core: A framework for sensor security application development," in *Mobile Adhoc and Sensor Systems (MASS), 2012 IEEE 9th International Conference on*, pp. 263–271, 2012.

[71] J. M. Prinsloo, C. L. Schulz, D. G. Kourie, W. H. M. Theunissen, T. Strauss, R. Van Den Heever, and S. Grobbelaar, "A Service Oriented Architecture for Wireless Sensor and Actor Network Applications," in *Proceedings of the 2006 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries*, SAICSIT '06, (Republic of South Africa), pp. 145–154, South African Institute for Computer Scientists and Information Technologists, 2006.

[72] R. Jurdak, A. G. Ruzzelli, A. Barbirato, and S. Boivineau, "Octopus: Monitoring, Visualization, and Control of Sensor Networks," *Wireless Communications and Mobile Computing*, vol. 11, no. 8, pp. 1073–1091, 2011.

[73] L. Ma, L. Wang, L. Shu, J. Zhao, S. Li, Z. Yuan, and N. Ding, "NetViewer: A Universal Visualization Tool for Wireless Sensor

Networks," in *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pp. 1–5, 2010.

[74] P. V. Biron and A. Malhotra, eds., *XML Schema Part 2: Datatypes*. W3C Recommendation, W3C, second ed., Oct. 2004.

[75] A. K. Dey, "Understanding and Using Context," *Personal Ubiquitous Comput.*, vol. 5, pp. 4–7, Jan. 2001.

[76] A. Taherkordi, R. Rouvoy, Q. Le-Trung, and F. Eliassen, "Supporting Lightweight Adaptations in Context-aware Wireless Sensor Networks," in *Proceedings of the 1st International Workshop on Context-Aware Middleware and Services: affiliated with the 4th International Conference on Communication System Software and Middleware (COMSWARE 2009)*, CAMS '09, (New York, NY, USA), pp. 43–48, ACM, 2009.

[77] X. Liu, "A Survey on Clustering Routing Protocols in Wireless Sensor Networks," *Sensors*, vol. 12, no. 8, pp. 11113–11153, 2012.

[78] M. Handy, M. Haase, and D. Timmermann, "Low Energy Adaptive Clustering Hierarchy with Deterministic Cluster-head Selection," in *Mobile and Wireless Communications Network, 2002. 4th International Workshop on*, pp. 368–372, IEEE, 2002.

[79] P. Ding, J. Holliday, and A. Celik, "Distributed Energy-efficient Hierarchical Clustering for Wireless Sensor Networks," in *Proceedings of the First IEEE international conference on Distributed Computing in Sensor Systems*, pp. 322–339, Springer-Verlag, 2005.

[80] "Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)."

[81] F. Stann, J. Heidemann, R. Shroff, and M. Z. Murtaza, "RBP: Robust Broadcast Propagation in Wireless Networks," in *Proceedings*

of the 4th international conference on Embedded networked sensor systems, pp. 85–98, ACM, 2006.

[82] G. Mainland, D. C. Parkes, and M. Welsh, "Decentralized, Adaptive Resource Allocation for Sensor Networks," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pp. 315–328, USENIX Association, 2005.

[83] T. N'Takpe and F. Suter, "Concurrent Scheduling of Parallel Task Graphs on Multi-clusters Using Constrained Resource Allocations," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pp. 1–8, May 2009.

[84] C.-L. Fok, G.-C. Roman, and C. Lu, "Adaptive Service Provisioning for Enhanced Energy Efficiency and Flexibility in Wireless Sensor Networks," *Science of Computer Programming*, vol. 78, no. 2, pp. 195 – 217, 2013. Coordination 2010.

[85] W. Li, F. C. Delicato, P. F. Pires, Y. C. Lee, A. Y. Zomaya, C. Miceli, and L. Pirmez, "Efficient Allocation of Resources in Multiple Heterogeneous Wireless Sensor Networks," *Journal of Parallel and Distributed Computing*, vol. 74, no. 1, pp. 1775 – 1788, 2014.

[86] S. Xiang, H. B. Lim, and K.-L. Tan, "Multiple Query Optimization for Wireless Sensor Networks," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pp. 1339 –1341, april 2007.

[87] N. Trigoni, Y. Yao, A. Demers, J. Gehrke, and R. Rajaraman, "Multi-query Optimization for Sensor Networks," in *Distributed Computing in Sensor Systems* (V. Prasanna, S. Iyengar, P. Spirakis, and M. Welsh, eds.), vol. 3560 of *Lecture Notes in Computer Science*, pp. 307–321, Springer Berlin Heidelberg, 2005.

[88] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection Tree Protocol," in *Proceedings of the 7th ACM Conference*

on Embedded Networked Sensor Systems, SenSys '09, (New York, NY, USA), pp. 1–14, ACM, 2009.

[89] "Dissemination of Small Values." http://www.tinyos.net/tinyos-2.x/doc/html/tep118.html.

[90] "Telos Revision B Datasheet." http://www.dtic.upf.edu/ bbellalt/telosb-datasheet.pdf.

[91] T. Instruments, "2.4 GHz IEEE 802.15.4 / ZigBee-Ready RF Transceiver (Rev. C)." http://www.ti.com/lit/ds/symlink/cc2420.pdf.

[92] D. Puccinelli, O. Gnawali, S. Yoon, S. Santini, U. Colesanti, S. Giordano, and L. Guibas, "The Impact of Network Topology on Collection Performance," in *Proceedings of the 8th European conference on Wireless sensor networks*, EWSN'11, (Berlin, Heidelberg), pp. 17–32, Springer-Verlag, 2011.

[93] "Canto Pairing function." http://en.wikipedia.org/wiki/Pairing_function.

[94] N. Kushalnagar, G. Montenegro, D. E. Culler, and J. W. Hui, "Transmission of Ipv6 Packets over IEEE 802.15. 4 Networks," 2007.

[95] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava, "Power Management in Energy Harvesting Sensor Networks," *ACM Trans. Embed. Comput. Syst.*, vol. 6, Sept. 2007.