Application-Driven Wireless Sensor Networks

Bruno Filipe Lopes Garcia Marques

Orientador: Manuel Alberto Pereira Ricardo (PhD)
Professor Associado com Agregação da Faculdade de Engenharia da Universidade do Porto

Programa Doutoral em Engenharia Electrotécnica e de Computadores

Dissertação submetida para satisfação parcial dos requisitos do grau de doutor em
Engenharia Electrotécnica e de Computadores

Janeiro, 2017

Aprovado em provas públicas pelo Júri

____ Presidente ____

Doutor José Alfredo Ribeiro da Silva Matos
Professor Catedrático da Faculdade de Engenharia da Universidade do Porto

____ Arguentes ____

Doutor Jorge Miguel Sá Silva
Professor Auxiliar da Faculdade de Ciências e Tecnologias da Universidade de Coimbra

Doutor Paulo Mateus Mendes
Professor Associado da Escola de Engenharia da Universidade do Minho

____ Vogais ____

Doutor Paulo José Lopes Machado Portugal
Professor Associado da Faculdade de Engenharia da Universidade do Porto

Doutor Ricardo Santos Morla
Professor Auxiliar da Faculdade de Engenharia da Universidade do Porto

Doutor Manuel Alberto Pereira Ricardo
Professor Associado com Agregação da Faculdade de Engenharia da Universidade do Porto

10 Janeiro 2017

This work was funded
by the Portuguese government through FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) grant «SFRH/BD/36221/2007»

To Cláudia, Tomás, and Carolina

Abstract

The growth of wireless networks has resulted in part from requirements for connecting people and advances in radio technologies. Recently there has been an increasing trend towards enabling the Internet-of-Things (IoT). Thousands of tiny devices interacting with their environments are being inter-networked and made accessible through the Internet. For that purpose, several communications protocols have been defined making use of the IEEE 802.15.4 Physical and MAC layers. The 6LoWPAN Network Layer adaptation protocol is an example which bridges the gap between low power devices and the IP world. Since its release, the design of routing protocols became increasingly important and the *IPv6 Routing Protocol for Low-Power and Lossy Networks* (RPL) emerged as the IETF proposed standard protocol for IPv6-based multi-hop Wireless Sensor Networks (WSN).

This thesis considers that the sensor nodes form a large IPv6 network making use of above technologies and protocols, and that the sensor nodes are enabled to run one or more applications. It is also assumed that the applications and the sensor nodes to which they are associated, are not always active, alternating between *active* and *inactive* states.

The thesis aims to design a new energy efficient communications solution for WSN by exploring the hypothesis that the network is aware of the traffic generated by the applications running in the sensor nodes. Therefore, the thesis provides two major contributions: 1) a cross-layer mechanism using application layer and network layer information to constrain *RPL-defined* routing trees (*RPL-BMARQ*); 2) an *Application-Driven WSN node synchronization* mechanism for *RPL-BMARQ*.

RPL-BMARQ is designed as an extension to the *RPL* routing protocol using information shared by the application and routing layers to construct *Directed Acyclic Graphs* (DAGs), allowing the nodes to select parents with respect to the applications they run. By jointly considering the neighbors of each node, the applications each node runs, and the forwarding capabilities of a node, we provide a communications solution which enables the data of every application and sensor node to be transferred, while keeping the overall energy consumed low by reducing the time the nodes are active and reducing the total number of multicast packets exchanged. Therefore, *RPL-BMARQ* helps reducing the network energy consumption since it restricts radio communication activities while maintaining throughput fairness and packet reception ratio high. The mechanism was evaluated using four scenarios with different network topologies and compared against *"standard RPL"*. The results obtained show that the mechanism enables lower energy consumption since the nodes are more often put a sleep, reducing the total number of packets exchanged, while maintaining fairness and query success rates hight.

The *Application-Driven WSN node synchronization* mechanism for *RPL-BMARQ* was designed to maintain the sensor nodes synchronized according to the duty cycle of the applications they run. The mechanism jointly uses cross-layer information and the *Exponentially Weighted*

Moving Average (EWMA) technique for calculating in run-time average network delays which are used to control the time the sensor nodes would sleep in the next cycle in order to wakeup just before the next activity period starts. This mechanism enables all the sensor nodes to go asleep and to wakeup in synchronism. The mechanism was theoretically evaluated and simulated, and the results obtained show that the synchronization mechanism works as previewed. The results also showed that, when designing WSN applications with this mechanism, the nodes not involved in communications are kept sleeping as much as possible, waking up when necessary and in synchronism.

In order to confirm the validity of the mechanisms designed, we also tested them in real environments where the results were confirmed.

Keywords: WSN, Application-Driven WSN, RPL, Node Synchronization.

Resumo

O crescimento das redes sem fios, resultou em parte dos requisitos para interligar pessoas e dos avanços das tecnologias rádio. Recentemente, tem havido uma tendência crescente no sentido de desenvolver a *Internet-das-Coisas*. Milhares de pequenos dispositivos que interagem com o ambiente têm sido interligados e acessíveis através da *Internet*. Para tal foram definidos vários protocolos de comunicação utilizando a camada física e a camada de acesso ao meio definidas pela norma *IEEE 802.15.4*. O protocolo de adaptação da camada de rede *6LoWPAN* é um exemplo e preenche o vazio existente entre os dispositivos de baixa potência e o mundo IP. Em combinação com o *6LoWPAN* surgiram novos protocolos de encaminhamento, entre os quais o protocolo de encaminhamento *RPL*, proposto pelo grupo *IETF* como um importante protocolo *"normalizado"* para as redes de sensores sem fios (*RSSF*) *"multi-hop IPv6"*.

Esta tese assume que os nós sensores formam uma grande rede *IPv6* e que utilizam as tecnologias e os protocolos acima indidados. Assume também que os nós sensores são capazes de executar uma ou mais aplicações e que as aplicações e os nós sensores às quais estão associadas não estão sempre ativos, alternando entre os estados *"ON"* e *"OFF"*.

A tese tem como objectivo a concepção de uma nova solução de comunicações energeticamente eficiente para *RSSF* e explora a hipótese de que a rede está ciente do tráfego gerado pelas aplicações em execução nos nós sensores. Assim, a tese apresenta duas contribuições principais: 1) um mecanismo intercamadas que utiliza informações da camada de aplicação e da camada de rede para restringir as árvores de encaminhamento (*RPL-BMARQ*); 2) um mecanismo de sincronização de nós sensores para as *RSSF* definidas por aplicações.

A solução *RPL-BMARQ* é desenhada como uma extensão ao o protocolo de encaminhamento *RPL* e utiliza informações compartilhadas pelas camadas de aplicação e de rede para construir *Grafos Acíclicos Direcionados* (DAGs), para permitir aos nós sensores selecionar os pais que correm as mesmas aplicações. Ao considerar em simultâneo os vizinhos de cada nó, as aplicações que cada nó executa e as capacidades de encaminhamento dos nós, desenvolvemos uma solução de comunicações que possibilita que os dados de cada aplicação e de cada nó sensor possam ser transferidos, mantendo o consumo energético total baixo através da redução do tempo em que os nós estão ativos e reduzindo o número total de pacotes *"multicast"* trocados. Portanto, a solução *RPL-BMARQ* ajuda a reduzir o consumo de energia nas RSSF já que restringe as atividades de comunicação rádio, mantém a equidade na transferência de pacotes nos nós, possuindo taxa de receção de pacotes elevada. O mecanismo foi avaliado em comparação com o protocolo *"normalizado RPL"*, utilizando quatro cenários com diferentes topologias de rede. Os resultados obtidos mostram que a solução proporciona um menor consumo energético, uma vez que os nós são mais frequentemente colocados a dormir, o que reduz o número total de pacotes trocados enquanto se mantém a equidade na transferência de pacotes e as taxas de sucesso de repostas.

O mecanismo de sincronização dos nós para a solução *RPL-BMARQ* foi desenvolvido para

manter os nós sensores sincronizados de acordo com o ciclo de trabalho das aplicações que eles executam. O mecanismo utiliza informações intercamadas em conjunção com a técnica da *Média Móvel Exponencialmente Ponderada* (EWMA) para calcular em tempo de execução, o tempo que os nós sensores deverão dormir até ao próximo ciclo de trabalho. Este mecanismo permite que os nós sensores acordem apenas o tempo necessário para realização das suas actividades. O mecanismo possibilita que todos os nós sensores possam adormecer e acordar em sincronismo. O mecanismo foi avaliado teoricamente e simulado e os resultados obtidos mostram que o mecanismo de sincronização funciona como o esperado.

Por forma a confirmar os resultados obtidos foram também feitas avaliações dos mecanismos desenvolvidos em ambiente real.

Keywords: RSSF, RSSF a pedido das aplicações, RPL, sincronização de nós.

Acknowledgements

First and foremost I would like to thank my advisor Prof. Manuel Ricardo. It was an honor to be his student. I appreciate all his contributions of time, ideas, and funding to make my work productive and stimulating. The joy and enthusiasm he has for research was contagious and motivational for me, even during tough times in the Ph.D. pursuit. Throughout all the difficulties I had, Prof. Manuel Ricardo was always understanding and guided me with patience, humanity, kindness and wisdom.

Thanks to Faculdade de Engenharia da Universidade do Porto for providing excellent education and accepting me as student.

I am also thankful to INESC TEC, where I have been extremely lucky to be accepted as collaborator for the past eight years and interact with such brilliant and insight full researchers. I truly enjoyed working with many people at INESC TEC and I would like to thank, in particular, all my colleagues in the Wireless Networks (WiN) research group at the Centre for Telecommunications and Multimedia (CTM), with whom many of the ideas of this work were discussed and improved due to their contribution.

I would like to thank Instituto Politécnico de Viseu (IPV) for funding and supporting, in part, my involvement in the PhD doctoral programme; I would also like to thank my close professional colleagues that, from the start, encouraged me and provided an extra motivation to pursue this objective.

As a personal note, I would like to leave a warm word of thanks to my family and friends, for their encouragement and moral support. To my mother and my father, for all their love. To my mother-in-law for her readiness and support, and to my friends André and Cátia for their encouragement.

Finally, my beloved wife, Cláudia, and my wonderful children Tomás and Carolina, deserve very special loving words, for being such incredible and kind human beings, who stood by my side and coped with my moody behavior during my research work, including the writing of my thesis in the last months and suffering with those absences during the weekends and evenings of those days. This journey would have been much harder without them in my life.

Bruno Marques

"A Person who
never made a mistake never tried anything new!"

Albert Einstein

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

6LoWPAN	IPv6 over Low Power Wireless Personal Network
AODV	Ad-hoc On-Demand Distance Vector Routing
APPID	Application ID
CSMA	Carrier Sense Multiple Access
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
DAG	Directed Acyclic Graph
DAO	Destination Advertisement Object
DIO	DODAG Information Object
DODAG	Destination-Oriented Directed Acyclic Graph
EED	End-to-End Delay
EWMA	Exponentially Weighted Moving Average
ETX	Expected Transmission Count
ETXOF	ETX Objective Function for RPL
FFD	Full Function Device
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IFS	Inter-Frame Space
IoT	Internet of Things
IP	Internet Protocol
IPHC	Internet Protocol Header Compression
LLN	Low Power, Lossy Network
LQI	Link Quality Indicator
LR-WPAN	Low Rate Wireless Personal Area Networks
MAC	Medium Access Control
MP2P	MultiPoint-to-Point
MPDU	MAC Protocol Data Unit
MRHOF	Minimum Rank with Hysteresis Objective Function
MTU	Maximum Transfer Unit
OF	Objective Function
OF0	Objective Function Zero
OSI	Open Systems Interconnection

P2MP	Point-to-Multipoint
P2P	Point-to-Point
PAN	Personal Area Network
PDU	Protocol Data Unit
PHY	Physical Layer
QSR	Query Success Ratio
RFC	Request for Comments
RFD	Reduced Function Device
RPL	Routing Protocol for Low Power and Lossy Networks
RPL-BMARQ	RPL By Multi-Application ReQuest
ROLL	Routing Over Low-power and Lossy networks
RSSI	Received Signal Strength Indicator
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UDGM	Unit Disk Graph Medium
WPAN	Wireless Personal Area Networks
WSN	Wireless Sensor Network

Chapter 1

Introduction

The growth of wireless networks has resulted in part from requirements for connecting people and advances in low power radio technologies. Wireless Personal Networks (WPANs) are an example of these networks, and Wireless Sensors Networks (WSN) [5, 6] are an example of WPANs. Sensing nodes measure a wide gamut of physical data [7] used for multiple applications. Whilst millions of sensing nodes may be deployed for current and future applications, they are likely to be associated in networks, interconnected or not to the Internet. Sensing nodes are energy constrained, and not every node is likely to reach the Internet gateway or other end point in a single hop; thereby routing strategies capable of finding the more energy-efficient paths are required.

1.1 Scope and Motivation

Recently there has been an increasing trend towards enabling the *Internet-of-Things* (*IoT*) [8]. As a subset, WSNs constituted by a large number of tiny devices interacting with their environments may be inter-networked together and accessible through the Internet. For that purpose, several communications protocols have been defined making use of the IEEE 802.15.4 Physical and MAC layers [9]. The 6LoWPAN Network Layer adaptation protocol [10] is an example which bridges the gap between low power devices and the IP world. Since its release, the design of routing protocols became increasingly important [11] and *RPL* [12] emerged as the IETF proposed standard protocol for IPv6-based multi-hop WSN.

Energy-efficiency has to be implemented also in large WSN which tend to be self-managed and self-configured. The behavior of these large systems need to be information-aware, where applications play the key role of waking up and enabling parts of the system. Our work is focused on *Application-Driven WSN* (ADWSN). We define ADWSN as a cross-layer solution aimed to help minimizing the energy consumed by a network of sensors executing a set of applications. We assume that sensors form a large network and that a sensor is enabled to run one or more

applications. We also assume that the applications and the nodes to which they are associated, are not always active, alternating between on and off states.

1.2 Problem Statement

In actual WSN every node can participate in route discovery and packet forwarding, independently of the applications in which the nodes participate. If we reduce the number of nodes participating in routing and forwarding functions, then energy will be saved. Assuming that, our hypothesis consists in associating these functions (routing and forwarding) mainly to the nodes supporting the application associated with the data to be transferred. By letting the nodes not associated to a given application to remain in a sleep status, we expect to have energy savings. Also, the nodes need to be awake in order to receive, send, and forward packets to the other nodes. Therefore, the nodes must be synchronized according to the application cycle they run in order to be awake almost at same time.

1.3 Objective

The main objective of this thesis is to develop a complete communications solution that can be used to interconnect a network of sensors. It is intended to be designed as an extension to the RPL routing protocol with the purpose of making the network aware of the traffic generated by applications. Sensors are assumed to form a large IPv6 network and a sensor is enabled to run one or more applications. It is also assumed that the applications and the nodes to which they are associated are not always active, alternating between *on* and *off states*. By jointly considering the neighbors of each node, the applications that each node runs, and the forwarding capabilities of a node, we developed a communications solution which enables the data of every application and node to be transferred while keeping the overall energy consumed by the network low.

We also assume that every node can participate in route discovery and packet forwarding. However, the nodes forwarding a given type of data, will be primarily selected from the set of nodes running the same application to which the data is associated.

It is unlikely that all the sensor nodes would join a network at the same time. Having the nodes active during all the time would deplete their batteries, so nodes have to go sleep and to wake up periodically. All the nodes have to be awake almost at same times in order to receive sink queries and to forward them to the other nodes. As a result, nodes must be synchronized according to the application cycle they run.

In order to pursue this these goals, the research efforts were divided in two particular objectives:

- Provide an Application-Driven mechanism in a form of a RPL extension to create and maintain DAGs according to the applications the sensor nodes run with minimal impact on network performance;

- Provide an Application-Driven WSN Node Synchronization mechanism to synchronize the nodes with respect to the applications they run and their application duty cycle, and foster energy efficiency.

1.4 Contributions

This thesis provides two main original contributions:

- *A novel mechanism using application-layer information to constrain RPL-defined routing trees* named *RPL-BMARQ*. It assumes that every node will primarily select its parent from a set of nodes running the same application and it mainly exchanges packets with neighbors running the same application, trying to avoid paths which may include nodes that do not run this application. For that purpose, the application layer of each node shares information with other layers of the communication stack. Parts of this contribution are also described in [13, 14, 15].

- *A novel Application-Driven WSN node synchronization mechanism for RPL-BMARQ*. Since the nodes may join the network at a non predictable and different times, they must share some kind of time reference which allow them to be synchronized with respect to the life cycle of the applications they run. Therefore, we propose an application-driven *synchronization mechanism* which helps the nodes running the same application to wakeup and to go asleep in a synchronized manner so they can successfully send, receive, and forward packets. Parts of this contribution are also described in [15].

1.5 Publications

- Bruno Marques, Manuel Ricardo, "**Application-Driven design to extend WSN lifetime**", in 1st Portuguese National Conference on Sensor Networks (CNRS2011), Coimbra, Portugal, March 4, 2011.

- Bruno Marques, Manuel Ricardo, "**Improving the energy efficiency of WSN by using application-layer topologies to constrain RPL-defined routing trees**", in 13th Annual Mediterranean Ad Hoc Networking Workshop, Piran, Slovenia, June 2-4, 2014.

- Bruno Marques, Manuel Ricardo, "**Energy-Efficient Node Selection in Application-Driven WSN**", *Wireless Networks, Springer Science, New York, ISSN: 1022-0038 (Print) 1572-8196 (Online), DOI: 10.1007/s11276-016-1194-2.*

- Bruno Marques, Manuel Ricardo, "**Synchronization of Application-Driven WSN**", submitted to *EURASIP Journal on Wireless Communications and Networking, Springer Open*. Expected to be published soon.

1.6 Document Structure

The structure of this thesis is as follows. Chapter 2 describes the WSN background related to this thesis; it also reviews the related work regarding Energy efficiency and node synchronization in Wireless Sensor Networks. Chapter 3 describes and evaluates the proposed Application-Driven mechanism (*RPL-BMARQ*). Chapter 4 describes and evaluates the proposed Application-Driven WSN node synchronization mechanism. Finally, Chapter 5 concludes the thesis and discusses future work.

Chapter 2

Energy Efficiency and Node Synchronization in Wireless Sensor Networks

In this chapter we present and discuss related work in the following main areas: WSN background, energy efficiency, time synchronization, and wakeup mechanisms for WSN. Since the contributions of this thesis must be supported by a WSN, Section 2.1 describes related concepts including technologies and protocols. Section 2.2 presents the state of the art on WSN energy efficiency. Section 2.3 presents the state of the art on time synchronization for WSN. Section 2.4 presents the state of the art on WSN nodes wakeup mechanism. Finally, this Chapter is summarized in Section 2.5.

2.1 WSN Background

In this section we discuss in detail some of the technologies and protocols used to achieve the goals of this thesis. Some of these technologies include the *Low-Power and Lossy Network* (*LLN*) concept, including the IEEE 802.15.4 standard, including *IPv6*, *6LoWPAN*, and routing protocols for IPv6 WSNs. We also discuss operating systems and simulation environments, with focus on *ContikiOS* [16] and *COOJA* [17], as well hardware platforms that researchers use to perform analysis and simulation of Wireless Sensor Networks, with focus on the *TelosB* platform [18].

2.1.1 Low-Power and Lossy Networks

Low-Power and Lossy Networks (*LLN*) are networks designed for long-lived applications. They consist of a number of low-power and low-cost nodes, and their communications are performed over multiple hops. *LLN* nodes may be powered with limited energy batteries what may lead to intermittent communications. *Wireless Sensor Networks* (*WSN*) are considered as a *LLN subset* [19]. Due to their capabilities LLNs are increasingly deployed in many application

domains, including industrial monitoring, factory, building and home automation, smart energy metering, and urban sensing [19, 20, 21]. Usually *LLN* implement the IEEE 802.15.4 standard [9], characterized by short range radio communications in the 2.4 GHz band and throughputs limited to 250 kbit/s.

2.1.2 Overview of the IEEE 802.15.4 Standard

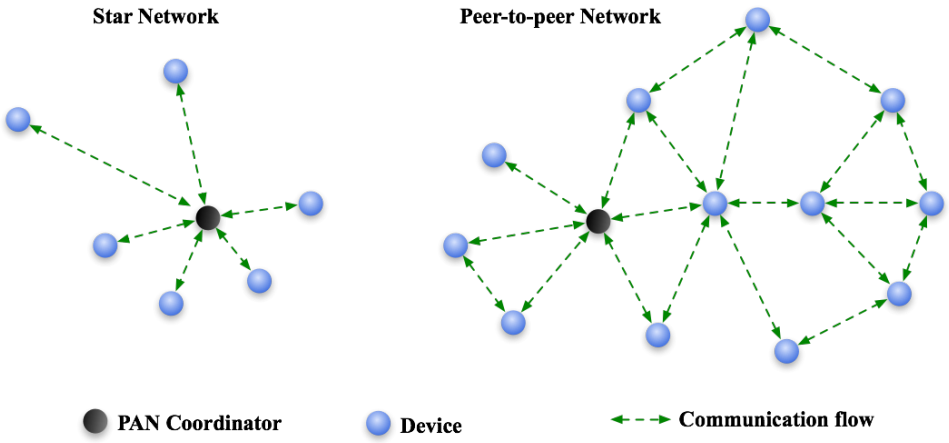

Figure 2.1: IEEE 802.15.4 Star and peer-to-peer networks

The IEEE 802.15.4 Standard [9] supports multiple network topologies, including star and peer-to-peer networks (see Fig. 2.1). Multiple address types, including 64-bit IEEE and short (16-bit network-assigned) are provided. Fig. 2.2 shows how IEEE 802.15.4 fits into ISO-OSI model.

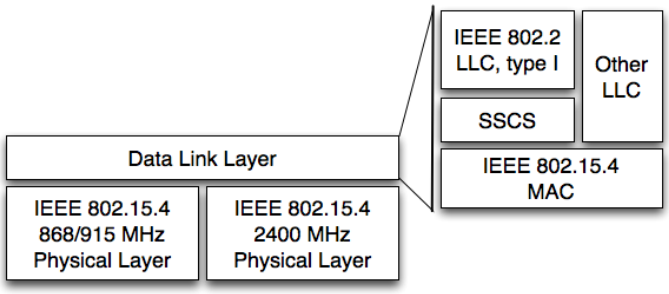

Figure 2.2: IEEE 802.15.4 in the ISO-OSI layered network model

IEEE 802.15.4 offers two PHY options that are combined with the MAC layer to enable a broad range of networking applications. Both PHYs are based on direct sequence spread spectrum (DSSS), and both share the same basic packet structure. The difference between the two PHYs is the frequency band they use (Fig. 2.3). The 2.4 GHz PHY specifies operation in the 2.4 GHz Industrial, Scientific, and Medical (ISM) band, while the 868/915 MHz PHY specifies operation in the 868 MHz band in Europe and 915 MHz ISM band in the United States. The 2.4 GHz PHY enables a transmission rate of about 250 kbit/s, while the 868/915 MHz PHY supports rates of 20 kbit/s and 40 kbit/s, respectively [9, 22].

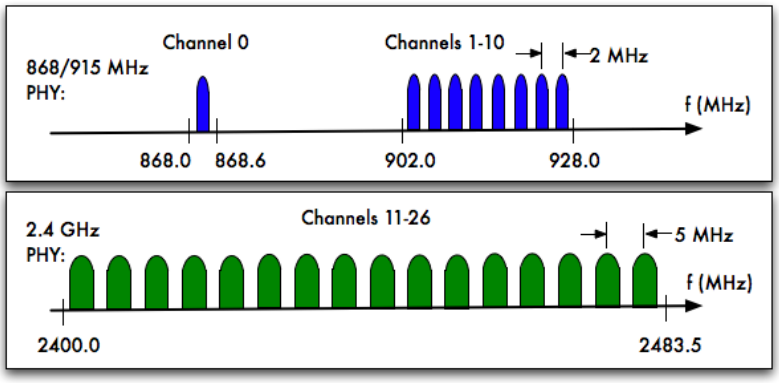

Figure 2.3: IEEE 802.15.4 channel structure

The 802.15.4 MAC frame structure is kept flexible in order to accommodate the needs of different applications and network topologies; its format is shown in Fig. 2.4. It consists of the MAC header (MHR), MAC service data unit (MSDU), and MAC footer (MFR). The first field of the MAC header is the frame control field; it indicates the type of MAC frame being transmitted, specifies the format of the address field, and controls the frame acknowledgment. The size of the address field may vary between 0 and 20 bytes. For instance, a data frame contains both source and destination information, while the acknowledgment frame does not contain source information - the acknowledgment frame uses the same sequence number (DSN) of the data or MAC command frame being acknowledged. Beacon frames contain only source address. Short 16-bit addresses or 64-bit IEEE addresses may be used. The payload field is variable in length, however the complete MAC frame may not exceed 127 bytes in length. The data contained in the payload depends on the frame type.

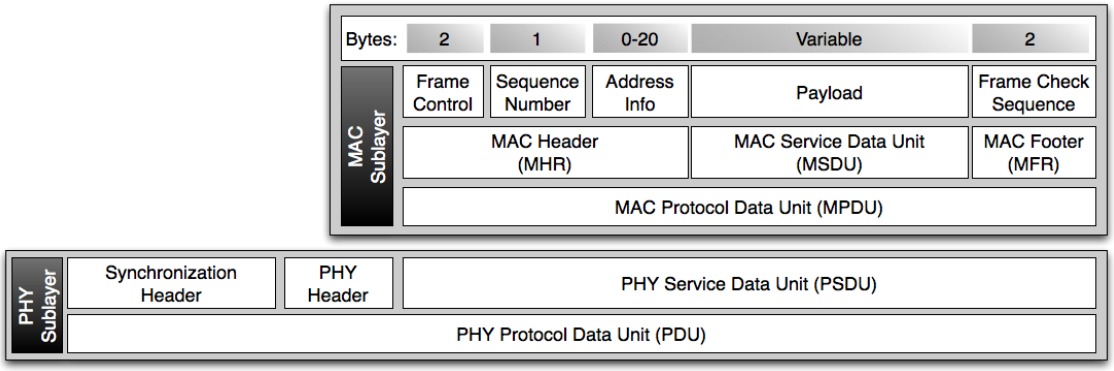

Figure 2.4: General IEEE 802.15.4 MAC frame format

A IEEE 802.15.4 MAC frame can be of four types: beacon frame, data frame, acknowledgment frame, and command frame. Only the data and beacon frames actually contain information sent by higher layers; the acknowledgment and MAC command frames are originated at the MAC layer and are used for MAC peer-to-peer communications. The other fields in a MAC frame are the sequence number and frame check sequence (FCS). The FCS helps to verify the integrity of the

MAC frame.

The MAC sub-layer provides two services to higher layers that can be accessed through two Service Access Points (SAP): the MAC Common Part Sublayer (MCPS-SAP), and the MAC Layer Management Entity (MLME-SAP), used for the MAC management services. Other MAC features can be found in [9, 22].

2.1.3 IPv6 in Low-Power Wireless Personal Area Network (6LoWPAN)

The *Internet Engineering Task Force* (IETF) 6LoWPAN addresses control and sensor networks working over wireless technologies [23], having designed a standard which defines how IP communications are performed over low-power WPAN [10] and it uses IPv6 [24]. 6LoWPAN [25] addresses the challenge of enabling wireless IPv6 communications over wireless sensor networks. A question which often arises is the following: is not the IP protocol too big for low-bandwidth networks? According to [26], resource conservation is the key factor for low-power wireless sensor applications. The resources that matter to achieve deployment ubiquity, long-lived power autonomy, and cost effective devices include: low protocol overhead over the wireless links, low program and data memory requirements, and low power usage in intermittent and infrequent sensor operation. On all these dimensions 6LoWPAN achieves efficiencies comparable to those obtained by non-IP based architectures such as ZigBee, while offering the benefits of end-to-end communication to a huge range of devices.

6LoWPAN are LLN which try to enable efficient transmission of IPv6 packets over 802.15.4 links [26]. IPv6 has an MTU of at least 1,280 bytes. The IEEE 802.15.4 frame length, in turn, is 127 bytes in order to ensure low packet error; in the worst case only 81 bytes of the frame may be used to transport data. These facts, combined with the low energy and low computing power of LLN nodes, present the following challenges to the deployment of IPv6 over IEEE 802.15.4:

1. A large IPv6 packet cannot be transmitted directly in an IEEE 802.15.4 frame, so segmentation and header compression techniques are required. The later is particularly important since the IPv6 headers could easily reach 80 bytes, leaving no space for data transmission;

2. The IEEE 802.15.4 links are prone to interference, failures, and may be asymmetric. These characteristics require the network layer to be responsive and adaptive while remaining energy efficient;

3. The IEEE 802.15.4 technology does not enable mesh, multi-hop, networks, so new networking functions are required.

In order to implement its functions, 6LoWPAN introduces an header between the 802.15.4 and the IPv6 headers. As it can be seen in Fig. 2.5 the 6LoWPAN header can be of multiple types, which can appear jointly or alone in the same frame, and may control the fields of the IPv6 header transmitted in the frame. As it can be observed in the figure, 6LoWPAN defines three headers [10]:

- **Dispatch header**: used to define the type of the next header in the frame;
- **Mesh header**: used to define the addresses of a packet transported over a multi-link network;
- **Fragmentation header**: used to fragment an IPv6 packet, and containing the information required to re-assemble the frame at the destination.

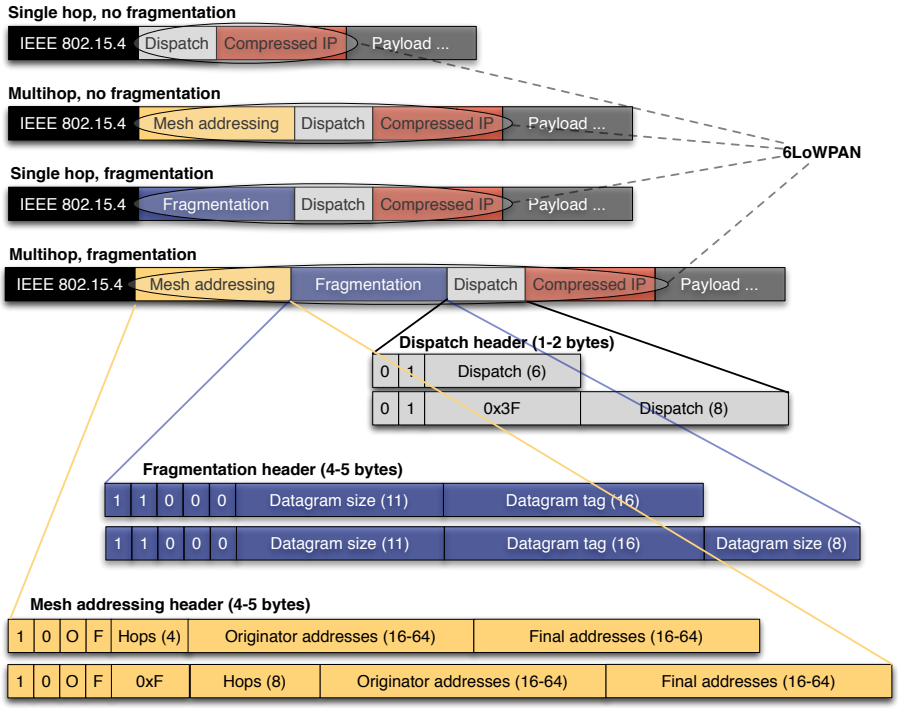

Figure 2.5: 6LoWPAN headers

6LoWPAN also uses header stacking to keep orthogonal concepts separate and supports compact headers by eliding headers that are unused. It defines a mesh addressing header to support layer two forwarding, a fragmentation header when the IPv6 datagram is too large to fit in a single 802.15.4 frame, and header compression to reduce IPv6 header overhead.

The 6LoWPAN Working Group has also optimized the existent IPv6 Neighbor Discovery procedure (ND) [27]. The ND protocol uses multicast communications and assumes that the link provides a single domain broadcast. Since the MAC layer of 6LoWPAN networks does not support multicast, broadcast communications are used to deliver multicast packets. This solution can cause some problems due to the lack of the acknowledgment service for the broadcast messages and moreover this type of transmission is usually more expensive. The challenges, applied by WG, remove the critical points and minimize the ND's reliance on multicast, obtaining as result the 6LoWPAN ND. This protocol uses Router Advertisement (RA) and Router Solicitation (RS) messages to give the nodes the possibility to find neighboring routers. The RS notifications are unique messages, sent on multicast mode. The main reasons are: 1) link-local IPv6 addresses are obtained from MAC addresses, so nodes do not have to perform address resolution; and 2)

IPv6 addresses, different from link layer addresses, are assumed not to be on-link and therefore communications with these nodes proceed to local routers.

2.1.4 Routing in IPv6 *LLN*s

Routing is the process of selecting paths in a network along which to send network traffic. In *LLN*s, routing directs packet forwarding, the transit of logically addressed packets from their source towards their ultimate destination through intermediate sensor nodes. The sensor nodes in the *LLN*s can also forward packets and perform routing, though they may suffer from its constraints [28]. The routing process usually directs forwarding on the basis of routing tables which maintain a record of the routes to various network destinations. Thus, constructing routing tables, which are held in the sensor nodes memory, is very important for efficient routing. Most routing algorithms use only one network path at a time, but multipath routing techniques enable the use of multiple alternative paths. There are four basic requirements for routing in *IPv6* based *LLN*s: (i) the sensor node should support sleep mode for considering battery saving; (ii) generated overhead on data packets should be low; (iii) routing overhead should be low; (iv) computation and memory requirements should be low. In the following we present and discuss the routing protocols mostly referenced in the literature that consider *LLN*s and their interconnection to the *Internet*.

2.1.4.1 6LoWPAN Mesh Routing

In 802.15.4 all nodes have a link to the coordinator node. Although IEEE 802.15.4 does not define mesh capabilities, 6LoWPAN are expected to implement mesh networks. A mesh network may employ one of two connection arrangements: full mesh topology or partial mesh topology. In the full mesh topology, each node is connected directly to each of the others. In the partial mesh topology, each node is connected directly to at least one other node, and there exists a path of direct connections between every pair of nodes. 6LoWPAN mesh networks may be characterized by: a) extension of network coverage without increasing transmit power or receive sensitivity; b) enhanced reliability via route redundancy, c) easier network configuration; d) longer device battery life due to fewer retransmissions.

According to [26], two important issues for IPv6 over 802.15.4 are how link-level factors inform routing, and at what layer datagram forwarding occurs. *6LoWPAN* in its role as an adaptation between the link layer and the network layer, introduces two types of mesh capabilities: the mesh under, and the route over. With mesh under capabilities, the network layer performs no IP routing; instead, the *6LoWPAN* adaptation layer seeks to mask the lack of a full broadcast at the physical level by transparently routing and forwarding packets within the *6LoWPAN* mesh network. By emulating a full broadcast link, it potentially provides compatibility with *IPv6* protocols that expect such communication behavior. Mesh topologies require forwarding over multiple radio hops, and link-local multicast must deliver packets to all nodes in the entire mesh network. The mechanisms to form, maintain, and diagnose *IP* routing must be recreated at the

link layer for the mesh under network to operate reliably. For mesh under strategies, 6LoWPAN has defined special routing protocols; one example is the *6LoWPAN Ad Hoc On-Demand Distance Vector Routing* (*LOAD*) [29]. LOAD is a simplified on-demand routing protocol based on *AODV* [30].

Route over, in alternative, performs routing at the *IP* layer, with each node serving as an *IP* router. We can view it as a collection of overlapping link-local scopes, with each link-local domain defined by the inherent radio connectivity. Unlike mesh under, route over supports layer three forwarding mechanisms that can utilize network-layer capabilities defined by *IP*, such as *ICMPv6*. Route over also lets *IP* routing protocols span different link technologies, enabling a better integration with other *IP* networks.

2.1.4.2 RPL

RPL [12] is a route over distance–vector (DV) and a source routing protocol that is designed to operate on top of several link layer mechanisms including IEEE 802.15.4 PHY and MAC layers. RPL was proposed by the IETF [31] ROLL working group [32] and is referred as *Routing Protocol for Low power and lossy networks*. Although RPL is still a RFC, it has gained a lot of maturity turning it as a promising standardized routing solution for Low Power and Lossy Networks. In fact, several research works have focused on the design and deployment of RPL protocol and real world implementations [33, 34, 35, 36, 37]. Moreover, [38] presents an overview on the *6LoWPAN* and *RPL* technologies. In that work, authors only provide a brief tutorial-like introduction of the RPL protocol. Olfa Gaddour et al. [39] present a comprehensive survey of the RPL protocol. A RPL specification is also presented in which the control messages, the *Directed Acyclic Graph* (DAG) construction, and the communication paradigms that RPL supports are described. Also how a RPL network is managed is presented. Based on their descriptions, in the following we provide an overview of RPL.

Design Objectives - RPL targets collection-based networks, where the nodes periodically send measurements to a collection point, as well as point-to-multipoint traffic from the central point to the devices inside LLNs. Point-to-Point traffic is also supported, and a key feature in RPL is that it represents a specific routing solution for LLNs [39]. Moreover, RPL has been specifically designed to meet the requirements of resource-constrained nodes. In particular, RPL-enabled LLNs take into account two main features: (1) the prospective data rate is typically low, and (2) communication is prone to high error rates, which results in low data throughput. A lossy link is not only characterized by a high Bit Error Rate (BER) but also the long inaccessibility time, which strongly impacts the routing protocol design. The protocol was designed to be highly adaptive to network conditions and to provide alternate routes, whenever default routes are inaccessible.

RPL is based on the topological concept of Directed Acyclic Graphs (DAGs), which defines a tree-like structure that specifies the default routes between nodes in the LLN. Moreover, the nodes are organized along a *Destination Oriented Directed Acyclic Graph* [40] (DODAG), normally rooted at a border router node or at a sink node, providing a default route to the Internet.

A network may consist of one or several DODAGs, which form together an RPL instance identified by a unique ID, called *RPLInstanceID*. A network may run multiple RPL instances concurrently and logically independent. An node may join multiple RPL instances, but must only belong to one DODAG within each instance. Fig. 2.6 shows an example of RPL instances with multiple DODAGs.

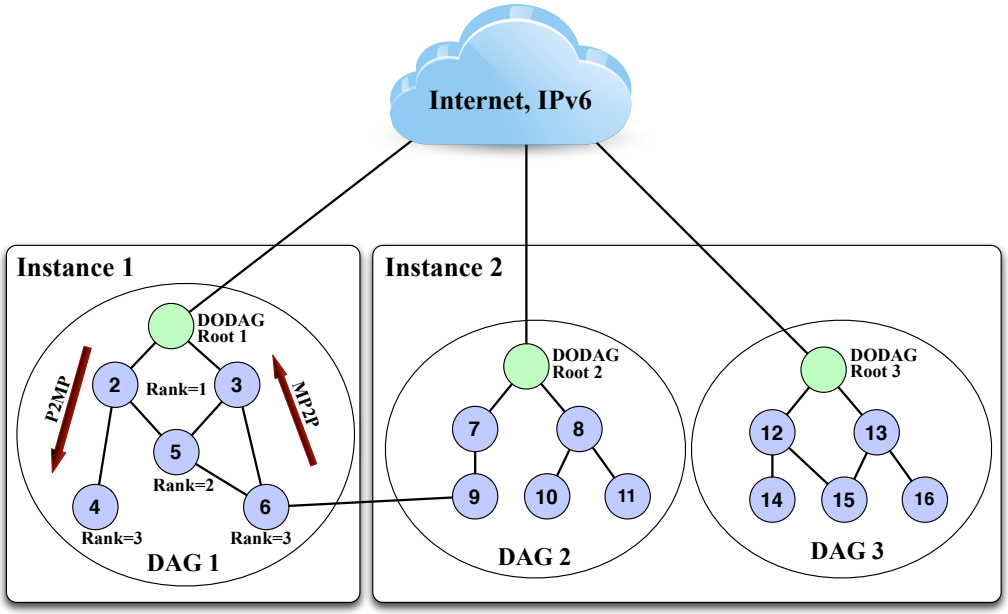

Figure 2.6: A RPL network with three DODAGs in two instances

One of the relevant features of RPL is that it combines both mesh and hierarchical topologies. An RPL-based network topology is inherently hierarchical as it forces underlying nodes to self-organize as one or several DODAGs, based on parent-to-child relationship. On the other hand, RPL supports the mesh topology as it allows routing through siblings instead of parents and children. This combination mesh/hierarchical provides a great flexibility in terms of routing and topology management, as described in the following [12]:

- *Auto-configuration:* as RPL is compliant with IPv6, the RPL-based LLN will benefit from basic IP routing characteristics mainly the dynamic discovery of network paths and destinations. This feature is guaranteed by the use of the Neighbor Discovery mechanisms.
- *Self-healing:* RPL proves its ability to adapt to logical network topology changes and node failures. In fact, links and nodes in LLNs are not stable and may vary frequently. RPL implements mechanisms that choose more than one parent per node in the DAG to eliminate/ decrease the risks of failure.
- *Loop avoidance and detection:* a DAG is acyclic by nature as a node in a DAG must have a higher rank than all of its parents. RPL includes reactive mechanisms in order to detect loops in case of topology change. In addition, RPL triggers recovery mechanisms (global and local repair) when the loops occur.

- *Independence and transparency:* as in the IP architecture, RPL is designed to operate over multiple link layers. RPL is able to operate in constrained networks, or in conjunction with highly constrained devices. Thus, RPL is then independent from data-link layer technologies.

- *Multiple edge routers:* it is possible to construct multiple DAGs in an RPL network and each DAG has a root. A node may belong to multiple instances, and may act different roles in each instance. Thus, the network will benefit from high availability and load balancing.

Network Model - RPL defines three types of nodes, which are:

- *Low Power and Lossy Border Routers (LBRs)*: it refers to the root of a DODAG that represents a collection point in the network and has the ability to construct a DAG. The LBR also acts as a gateway (or edge router) between the Internet and the LLN.

- *Router*: it refers to a device that can forward and generate traffic. Such a router does not have the ability to create a new DAG, but associate to an existing one.

- *Host*: it refers to an end-device that is capable of generating data traffic, but is not able to forward traffic.

The basic topological component in RPL is the *DODAG*, a *Destination Oriented DAG*, rooted in a special node called DODAG root, as illustrated in Fig. 2.6. The DODAG root has the following properties: (i) it typically acts as an LBR, (ii) it represents the data sink within the directed acyclic graph, (iii) it is typically the final destination node in the DODAG, since it acts as a common transit point that bridges the LLN with IPv6 networks, (iv) it has the ability to generate a new DODAG that trickles downward to leaf nodes.

Each node in the DODAG is assigned a rank. The rank of a node is defined in [12] as *"the node's individual position relative to other nodes with respect to a DODAG root"*. It is an integer number that represents the location of a node within the DODAG. The rank strictly increases in the downstream direction of the DAG, and strictly decreases in the up-stream direction. In other words, nodes on top of the hierarchy receive smaller ranks than those in the bottom and the smallest rank is assigned to the DODAG root (see Fig. 2.6).

The architecture of a DODAG is similar to a cluster-tree topology where all the traffic is collected in the root. However, the DODAG architecture differs from the cluster-tree in the sense that a node can be associated not only to its parent (with higher rank), but also to other sibling nodes (with equal ranks). The rank is used in RPL to avoid and detect routing loops, and allows nodes to distinguish between their parents and siblings in the DODAG. In fact, RPL enables nodes to store a list of candidate parents and siblings that can be used if the currently selected parent loses its routing ability.

In the construction process of network topology, each router identifies a stable set of parents on a path towards the DODAG root, and associates itself to a preferred parent, which is selected based on the Objective Function. The Objective Function defines how RPL nodes translate one

or more metrics into ranks, and how to select and optimize routes in a DODAG. It is responsible for rank computation based on specific routing metrics like delay, link quality, and connectivity, and specifying routing constraints and optimization objectives. The design of efficient Objective Functions is still an open research issue. A couple of RFCs have been proposed. In [41], the RFC proposes the Minimum Rank with Hysteresis Objective Function, making use of use the Expected Number of Transmission (ETX) required to successfully transmit a packet on the link as the path selection criteria in RPL routing. The route from a particular node to the DODAG root represents the path that minimizes the sum of ETX from source to the DODAG root. In [42], the RFC proposes Objective Function 0 (OF0), which is only based on the abstract information carried in an RPL packet, such as Rank. OF0 is agnostic to link layer metrics, such as ETX, and its goal is to promote connectivity between network nodes.

Protocol Specification - In the following, the main mechanisms and features provided by RPL are presented, as defined in the IETF RFCs.

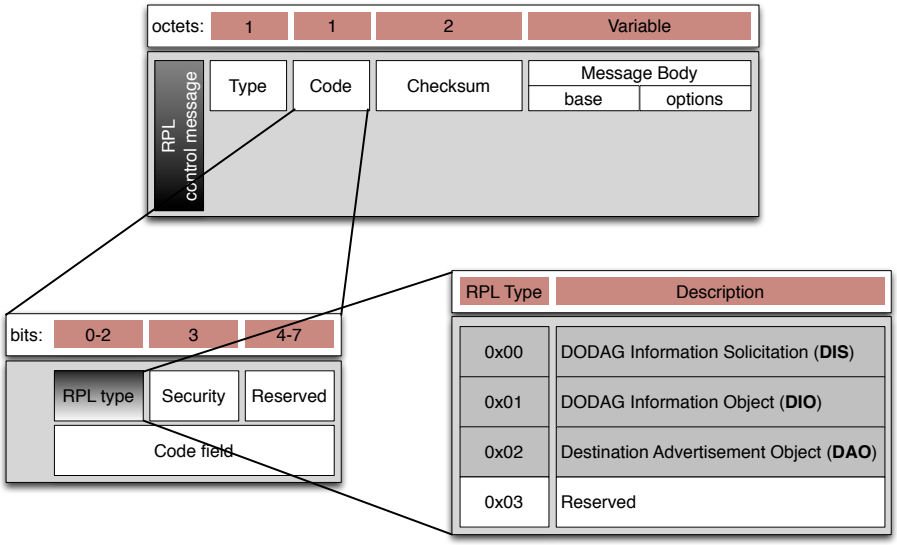

Figure 2.7: A RPL control message

1. **RPL Control Messages**: RPL messages are specified as a new type of ICMPv6 control messages, whose structure is depicted from Fig. 2.7. The RPL control message is composed of: i) an ICMPv6 header, which consists of three fields: *Type*, *Code* and *Checksum*; ii) a message body comprising a message base and a number of options.

   The *Type* field specifies the type of the ICMPv6 control message prospectively set to 155 in case of RPL. The *Code* field identifies the type of RPL control message. For this, four codes are currently defined:

   DODAG Information Solicitation (DIS): the DIS message is mapped to *0x00*, and is used to solicit a DODAG Information Object (DIO) from an RPL node. The DIS may

be used to probe neighbor nodes in adjacent DODAGs. The current DIS message format contains non-specified flags and fields for future use.

DODAG Information Object (DIO): the DIO message is mapped to *0x01*, and is issued by the DODAG root to construct a new DAG and then sent in multicast through the DODAG structure. The DIO message carries relevant network information that allows a node to discover a RPL instance, learn its configuration parameters, select a DODAG parent set, and maintain the DODAG. The format of the DIO Base Object is presented in Fig. 2.8. The main DIO Base Object fields are: (i) RPLInstanceID, is an 8 bit information initiated by the DODAG root that indicates the ID of the RPL instance that the DODAG is part of, (ii) Version Number, indicates the version number of a DODAG that is typically incremented upon each network information update, and helps maintaining all nodes synchronized with new updates, (iii) Rank, a 16 bit field that specifies the rank of the node sending the DIO message, (vi) Destination Advertisement Trigger Sequence Number (DTSN) is an 8 bit flag that is used to maintain downward routes, (v) Grounded (G) is a flag indicating whether the current DODAG satisfies the application-defined objective, (vi) Mode of Operation (MOP) identifies the mode of operation of the RPL instance set by the DODAG root. Four operation modes have been defined and differ in terms of whether they support downward routes maintenance and multicast or not. Upward routes are supported by default. Any node joining the DODAG must be able to cope with the MOP to participate as a router, otherwise it will be admitted as a leaf node, (vii) DODAGPreference (Prf) is a 3 bit field that specifies the preference degree of the current DODAG root as compared to other DODAG roots. It ranges from *0x00* (default value) for the least preferred degree, to *0x07* for the most preferred degree, (viii) DODAGID is a 128 bit IPv6 address set by a DODAG root, which uniquely identifies a DODAG. Finally, DIO Base Object may also contain an Option field.

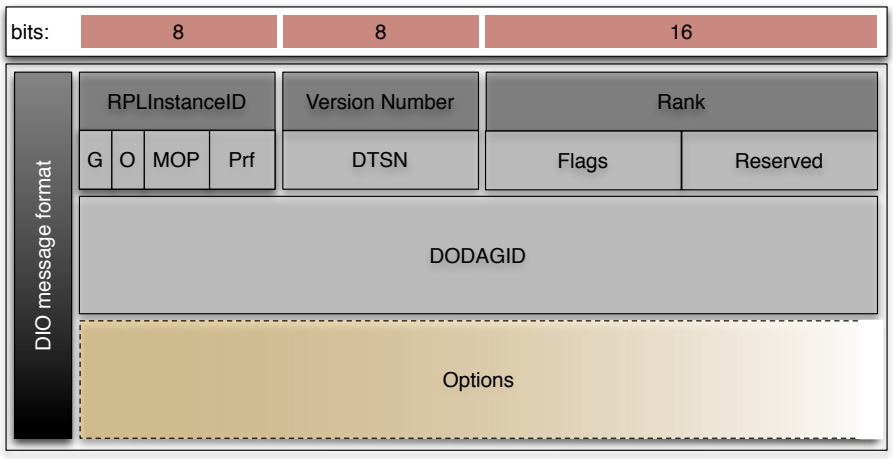

Figure 2.8: The DIO message format

Destination Advertisement Object (DAO): the DAO message is mapped to *0x02*, and is used to propagate reverse route information to record the nodes visited along the upward

path. DAO messages are sent by each node, other than the DODAG root, to populate the routing tables with prefixes of their children and to advertise their addresses and prefixes to their parents. After passing this DAO message through the path from a particular node to the DODAG root through the default DAG routes, a complete path between the DODAG root and the node is established. Fig. 2.9 shows the format of the DAO Base Object.

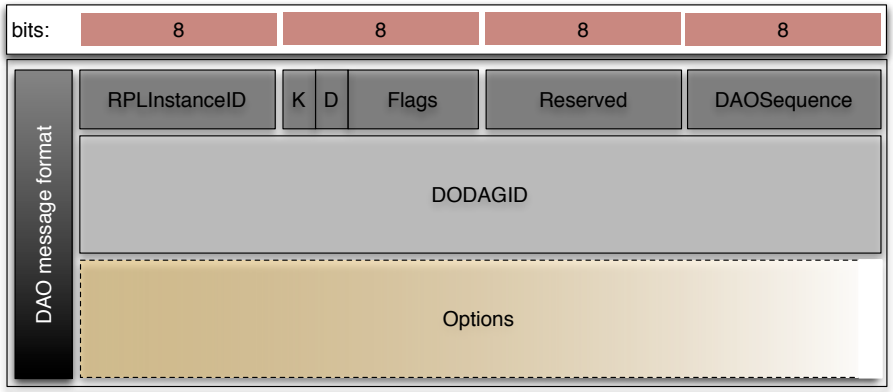

Figure 2.9: The DAO message format

As shown in the Fig. 2.9, the main DAO message fields are: (i) RPLInstanceID, is an 8 bit information indicates the ID of the RPL instance as learned from the DIO, (ii) K flag that indicates whether and acknowledgment is required or not in response to a DAO message, (iii) DAO-Sequence is a sequence number incremented at each DAO message, (iv) DODAGID is a 128 bit field set by a DODAG root which identifies a DODAG. This field is present only when flag D is set to 1.

Destination Advertisement Object (DAO-ACK): the DAO-ACK message is sent as a unicast packet by a DAO recipient (a DAO parent or DODAG root) in response to a unicast DAO message. It carries information about RPLInstanceID, DAOSequence, and Status, which indicate the completion. Status code are still not clearly defined, but codes greater than 128 mean a rejection and that a node should select an alternate parent.

2. **DODAG construction**: the DODAG construction is based on the Neighbor Discovery (ND) process, which consists in two main operations (1) broadcast transmission of DIO control messages issued by the DODAG root to build routes in the downward direction from the root down to client nodes, (2) unicast of DAO control messages issued by client nodes and sent up to the DODAG root to build routes in the upward direction.

   In order to construct a new DODAG, the DODAG root broadcasts a DIO message to announce its DODAGID, its Rank information to allow nodes to determine their positions in the DODAG, and the Objective Function identified by the Objective Code Point (OCP) within the DIO Configuration option fields. This message will be received by a client node which can be either a node willing to join or an already joined node.

When a node willing to join the DODAG receives the DIO message, it (i) adds the DIO sender address to its parent list, (ii) computes its rank according to the Objective Function specified in the OCP filed, such that the node's rank is greater than that of each of its parents, and (iii) forward the DIO message with the updated rank information. The client node chooses the most preferred parent among the list of its parents as the default node through which inward traffic is forwarded.

When a node already associated with DODAG receives another DIO message, it can proceed in three different ways: (i) discard the DIO message according to some criteria specified by RPL, (ii) process the DIO message to either maintain its location in an existing DODAG or (iii) improve its location by getting a lower rank in the DODAG based on computing the path cost specified by the Objective Function. Whenever a node changes its rank, it must discard all nodes in the parents' list whose ranks are smaller than the new computed node's rank to avoid routing loops. Fig. 2.10 shows the flowchart summarizing the operation of a router in a DODAG.

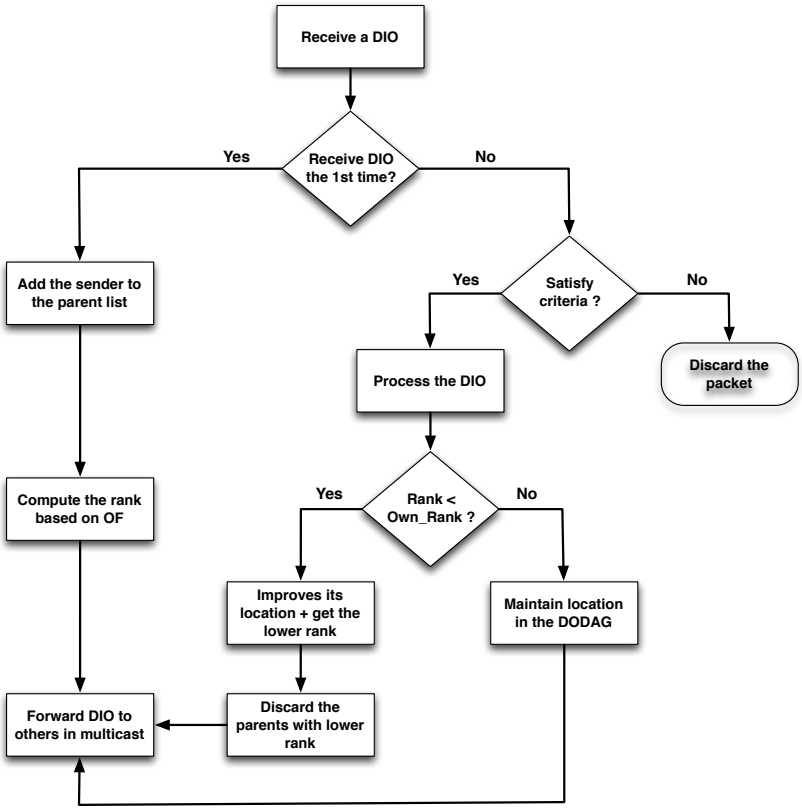

Figure 2.10: Operation of a router in a DODAG

After the construction of the DODAG, each client node would have a default upward route through which it can transmit its inward traffic at the destination of the DODAG root. Obviously, the default route is formed by the most preferred parent of each node.

If the Mode of Operation flag in the DIO Base Object is different from zero, downward

routes from the root to nodes are supported and have to be maintained. In this case, each client node must send a unicast DAO control message to determine the reverse route information. When traveling back to the DODAG root, visited nodes are recorded in the packet along the upward route, and complete route is then established between the DODAG root and the client node. RPL specifies two modes of operations to maintain downward routes in an RPL instance:

Storing Mode: in the storing mode, a DAO message is sent in unicast by the child to the selected parent, which is able to store DAO messages received by its children before sending the new DAO message with aggregate reachability information to its parent. The storing mode can enable or disable multicast mode.

Non-Storing Mode: in the non-storing mode, the DAO message is sent in unicast to the DODAG root, thus, intermediate parents do not store DAO messages, but only insert their own addresses to the reverse route stack in the received DAO message, then forwards it to its parent. To maintain the DODAG, each node periodically generates DIO messages triggered by a trickle [43] timer. The key idea of the trickle timer technique is to optimize the message transmission frequency based on network conditions. In a nutshell, the frequency is increased whenever an inconsistent network management information is received for faster recovery from a potential failure, and decreased in the opposite case. The timer duration increases exponentially whenever the timer fires. Initially, the timer duration is set to I_{min}, which will be doubled $I^{doubling}$ times until it reaches the maximum value $I_{max} = I_{min} \cdot I^{doubling}$. For any detected change in the DODAG (e.g. unreachable parent, new parent selection, new DODAG Sequence Number, routing loop, etc.), the trickle timer is reset to I_{min}, prescribed in DIO messages.

3. **Communication Paradigms**: RPL supports three communication paradigms: (i) Multi-Point-to-Point (MP2P) (or many-to-one); (ii) Point-To-Multi-Point (P2MP) (or one-to-many); (iii) Point-To-Point (P2P) (or one-to-one), which are detailed in the following.

   Multi-Point-to-Point: RPL provides support for multipoint-to-point traffic which pertains to data collection traffic forwarded in the upward route direction from multiple nodes towards the DODAG root. The data collection traffic is referred to as inward unicast traffic. The MP2P traffic is the main traffic flow in most of LLN applications [44, 45, 46, 47]. The destinations of MP2P are mainly border routers that play important roles in the network and provide an interface for connectivity with the Internet. RPL supports MP2P traffic such that destinations can be reached via DODAG roots. Once the DODAG is constructed, it is used to build the upward routes from routers to the root. The default routes from nodes to the roots are established through the preferred parents chain. The DAG root can insert in its DIO messages the destination prefixes which may also be included by DIOs generated by the routers through the LLN, to which it can provide connectivity. The main advantage of MP2P traffic in RPL is that it is supported with little routing state as the node should only store the destination that leads to the DAG root.

Point-to-Multipoint: RPL also defines the Point-to-Multipoint Operation, which represents the traffic transmitted in the downward route direction from the root down to multiple nodes. This configuration traffic is commonly known as outward unicast traffic, and it is essential for some LLN applications such as *Home Automation* [44], and *Industrial Automation* [45]. To support P2MP traffic, RPL uses a destination advertisement mechanism, which supplies down routes to routers until reaching the destination. To install downward routes, the routers send unicast DAO messages to their parents or to the DAG root. The DAO messages contain the prefixes within the network, and advertise the routes for each destination. Each intermediate router that forwards a DAO message towards the root adds its address to a reverse routing route in the DAO message, thus providing the source with the ability of performing source routing for reaching the child nodes in the network.

Point-to-Point: RPL provides mechanisms for point-to-point (P2P) routing between any two nodes in the DODAG. In order to support P2P traffic in a RPL network, a LBR must be able to transit packets at which point it is source routed to the destination. Two cases arise: (i) if the destination node is co-located with the sender node in the same transmission range, then the latter directly sends the message to the destination without passing it through its parent; (ii) else, the P2P mechanism would depend on whether the network is pre-configured as storing or non-storing mode. In the non-storing mode, routers do not store any information about downward routes (no information about their descendants) and only the root possesses such information. In this case, any packet must be first sent through the DODAG upward routes to the root, which will forward it to its destination. In the storing mode, routers locally store the routing information about downward routes. If the destination is a descendant of the router, then it forwards the message down to the closest router to the destination. If the destination is not a descendent, the message is forwarded to the parent node, which will apply the same rules to send the packet to its destination. Therefore, the packet will be forwarded in the uplink direction of the tree from child to parent until reaching the router that is the first ancestor of both the source and destination nodes.

4. **Multicast Routing**: is supported by RPL only in the storing mode, when the (MOP) field in the DIO control message is set to 0x03.

   RPL relies on the Multicast Listener Discovery (MLD) group registration protocol [48] for supporting multicast routing. In fact, it maps MLD queries into RPL DAO messages by transporting a multicast group in DAO target option. This mapping enables a DODAG root to act as a multicast router as if the listener were directly attached to it. Nodes that support the multicast operation can join the network as routers, but those that do not support multicast can only join as leaf nodes.

   If a listener is not an RPL node, then it uses the typical MLD protocol to register to a multicast group. If this listener is attached to an RPL router while multicast is supported, then the RPL router map the MLD queries into a DAO message for the registration of the

requesting node. If the listener is already RPL-enabled, then DAO message are used by default for group registration. MLD requests are then transported as DAO messages within the DODAG recursively between child and its parent up to the DODAG root.

Multicast routing information are located at each router on the way from the nodes to the DODAG root, enabling the root to send a multicast packet to all its children that had issued a DAO message requesting for that multicast group, as well as all the attached nodes that registered over this multicast group [12].

When a node sends a multicast packet inside the DODAG, the packet is forwarded to the preferred parent, by default. If the transmission fails, then the packet will be routed to the alternate parents in the DODAG. The packet is also duplicated to all the registered children, except for the one that passed the packet.

The multicast operation is then centralized in the DODAG root which acts as an automatic proxy point for the RPL network, and as source towards the non-RPL domain for all multicast flows started in the RPL domain.

Network Management - fault-tolerance, QoS-aware routing and security are RPL's mechanisms for network management. In the following we present them.

1. **Fault-Tolerance**: RPL is a self-healing routing and topology control protocol. In fact, it presents mechanisms for: i) DODAG repair operation, triggered when an inconsistency is detected in the DODAG; ii) loop detection and avoidance mechanisms to avoid loops in routing.

   DODAG Repair: repair mechanisms are of paramount importance for a routing protocol to dynamically update routing decisions and adapt the network topology to potential node/link failure. To that end, RPL supports two complementary repair mechanisms: (i) global repair and (ii) local repair. When a node detects a network inconsistency (e.g. a link between two nodes fails or a local loop is detected), it triggers a local repair operation. It consists in urgently finding a backup path without trying to repair the whole DODAG. This alternate recovery path may not be an optimal path.

   If local repairs are not efficient for network recovery due to several inconsistencies, the DODAG root may trigger a global repair operation and then it increments the DODAG version number and initiates a new DODAG version. The global repair operation leads to a fundamental reconstruction of the network topology. Nodes in the new DODAG version can choose a new position whose rank is neither constrained by nor dependent on their previous rank within the old DODAG version.

   Loop Avoidance and Detection: Loops are a common undesirable problem in distance–vector routing protocols. To overcome it, RPL relies on rank-based and path-validation mechanisms for loop avoidance and loop detection. The loop avoidance and detection mechanisms are different from those defined in traditional IP networks [38]: in fact, in

LLNs, the overreaction to loop detection is not recommended as opposed to IP networks where fast reaction is a must, since the prospective traffic in LLNs is very low, thus limiting the influence of potential loops.

A loop may occur when a node that loses all its parents, for some reasons, and attaches to another node in its own sub-DODAG. This may happen in particular when DIO messages are lost, and referred to as DIO loops. Several rules have been defined to avoid loops. The *max_depth* rule imposes that: a node cannot select a parent whose rank is higher than the node minimum rank plus *DAGMAXRankIncrease*. This rule does not prevent loops from occurring but avoid the *count-to-infinity problem* when a loop is formed. For instance, considering the network in Fig. 2.6, assume that the link between node 2 and the root node was broken, and node 2 selects node 6 as parent. This situation results in a loop as node 2 has no means to know that node 6 belongs to its sub-DODAG. A local repair operation will detect and repair the loop after a few network updates, which will end by dissociating node 2 from node 6. The repair operation works as follows: if *DAGMAXRankIncrease* = 5, this means that node 2 can join any node with a rank $(1 + 5) = 6$. If node 2 selects node 6 as a parent, its new rank is increased to 4. Then, node 5 updates in turn its rank to 5, node 6 updates its rank to 6, and node 2 updates again its rank to 7, which exceed the maximum allowed rank, that is 7. The loop is then detected, and thus avoided by breaking the child-parent relation between node 2 and node 6. RPL also prevents nodes looking for alternate parents to increase their ranks by selecting deeper parents, since this would very likely results in loops in the network.

Another way to avoid loops is that a node may poison its sub-DAG by advertising a rank of *INFINITE_RANK* without having to use *DAGMaxRankIncrease*. In addition to DIO loops, DAO loops may occur when a node fails to inform its parent that a destination is no longer reachable. In other words, the parent maintains a route installed towards a destination based on old DAO messages from a child, but the child is not able to update its parent about the non availability of that route, due to DAO message loss. In this case, if the child wants to send a packet to that destination, its non updated parent will keep sending back the packet to the child, thus forming a loop. The use of acknowledgment of DAO messages represents a fundamental solution to avoid DAO loops.

Loops are often hard to avoid. Thus, RPL specifies loop detection mechanisms to resolve them when they do occur. Detection mechanisms rely on the concept of data path validation, which consists in carrying control data in data packets to ensure that a packet is moving in forward direction and not experimenting any loop. Control data are flags making part of packet headers. For instance, the current RPL version allows one-hop sibling path, which means that a packet can be forwarded to a sibling only once along its path to the destination. The packet is then dropped in the second attempt to forward the packet to a sibling of another node, which can be easily encoded with a flag in packet header.

2. **QoS-Aware Routing**: RPL is a QoS-aware and constrained-based routing protocol. QoS-

aware routing means that RPL is able to provide different levels of QoS based on the underlying QoS metric ruling routing decisions. Constraint-based routing means that RPL defines constraints that reduce the search space for possible path satisfying QoS requirements. QoS metrics and QoS constraints are typically different from each other. A metric is a quantitative value that helps to find the best path satisfying an Objective Function. For instance, an Objective Function based on the delay metric would be to find the path that minimizes the end-to-end delay from the source to the destination. On the other hand, a constraint is used to include or eliminate links or nodes that do not respect specific criteria [38]. For example, the Objective Function would recommend to prune/avoid links with poor quality. A routing metric or constraint can be additive or multiplicative, or selects a path that contains a maximum or a minimum value. In addition, routing metric can be either local, when it does not propagate along the DODAG, in contrast to global routing metrics which should be propagated.

In [49], the ROLL working group specified a set of links and nodes link's and node's routing metrics and constraints that are suitable to LLNs and recommended for the RPL routing protocol. RPL defines in the DIO control message a common optional header for metrics and constraints objects, called DAG Metric Container object, with several flags. Flags are used to specify the nature and features of routing objects, such as whether it represents a metric or constraints, whether it is local or global, whether a metric is additive or others, etc. The main routing metric and constraint objects are summarized as follows.

Node State and Attribute (NSA): reports information on node state and attributes such as CPU overload, available memory. The ROLL working group decided to set a 1 bit flag when a node faces a congestion situation, assessed according to a local policy. If the flag is set, traffic will be re-routed to avoid passing through the congested node;

Node Energy: is used as a constraint to avoid using nodes with low power level. In [49], the Node Energy Object can be described by any combination of the following indicators: (1) the node power mode, encoded by 2 bit flag indicating the type of the node's power sources: main-powered, battery-powered, or scavenger (solar panel, mechanical, etc.), (2) the estimated remaining lifetime, which provides an estimation of the remaining power-level for nodes operating with batteries and scavengers, (3) other power-related metrics to be defined in the future;

Hop Count: reports the number of hops crossed along the path between a node and the destination;

Link Throughput: reports the range of throughput that the link can handle, in addition to the current link throughput. It can be used as a metric or constraint. When used as a metric, it may be considered as additive, or it may report a maximum or a minimum value;

Link Latency: is used to report the path latency. The latency of the path is expressed as the sum of all latencies, or can be mapped to the maximum/minimum latency along the

path. It can also be used as a constraint (e.g. pruning all links with a latency higher than a certain threshold);

Link Reliability: specifies the link reliability level, which can be expressed in several ways such as packet reception ratio, bit error ratio, mean time between failures, and others. In [49], two links reliability metrics have been defined: (1) the Link Quality Level (LQL) object, which quantifies the link reliability using a discrete value, from 0 to 7 where 0 indicates that the link quality level is unknown and 1 reports the highest link quality level. The mechanisms specifying how to compute LQL has still not been defined and are implementation specific. (2) *Expected Transmission count Metric* (ETX), which provides an estimation of the number of transmission a node has to make along the path to the destination to deliver a packet. It is typically estimated as $1/(PPR_{up} \cdot PPR_{down})$ that is the inverse of the product of packet reception ratio (PRR) in upstream and downstream directions;

Link Color: this constraint allows to assigning 10 bit encoded color to links, where the meaning of each color is implementation-specific. This administrative static link constraint is used to avoid or attract specific links for specific traffic types. For instance, it is possible to assign a blue color for links supporting encryption. This color object can be used to define specific Objective Function, such as selecting blue colored paths, or paths with maximum number of blue colored links, if encryption is an important criteria in the routing policy.

2.1.4.3 ZigBee IPv6-based Stack

In the recent past, the ZigBee Alliance introduced a communication stack for wireless sensor networks meeting the typical requirements of low data-rate lossy links interconnecting low-power devices. Three device types are defined in ZigBee: ZigBee coordinator, ZigBee routers, and ZigBee end devices. An reduced function device (RFD) can only be a ZigBee end device, whereas a full function device (FFD) can be either a ZigBee coordinator or ZigBee router. The ZigBee coordinator is responsible for starting a new network. ZigBee coordinator and routers are "routing capable", while the ZigBee end devices cannot participate in routing and have to rely on their corresponding ZigBee parent routers for that function.

Every node in a ZigBee network has two addresses: a 16 bit short network address, and a 64 bit IEEE extended address. The 16 bit network address is assigned to each node dynamically by its parent coordinator/router upon joining the network. This address is the only address that is used for routing and data transmission. There are schemes for assigning the 16 bit short network addresses in ZigBee mesh networks: the static address allocation, and the tree address allocation. In both schemes the parent nodes assign an address "block" to their child router which, in turn, uses it for its decedents. The mesh coordinator/router is responsible to maintain the free address spaces, the address block size, and the net address to be assigned.

Based on IEEE 802.15.4, ZigBee [50] specifies the application and network layers. The application layer framework consists of a set of application objects. The ZigBee network layer defines how the network is formed and how the network address is assigned to each participating

node. Two routing schemes are available in ZigBee networks: mesh routing, and tree routing. The mesh routing is similar to the AODV [30] routing algorithm, while the tree routing scheme resembles the cluster tree routing algorithm described in [51]. In ZigBee mesh routing, a route request (RREQ) is broadcasted on-demand when data is to be transmitted to a destination of an unknown path. Routes are constructed based on the route replies, received both intermediate nodes or the destination node, and a route error (RERR) message is transmitted when a path cannot be found. The route repair mechanism repairs invalid routes when a previous route can not be found. Since only coordinators/routers (FFDs) can actively participate in mesh routing, the end devices (RFDs) have to rely exclusively on their parent nodes to perform mesh routing on their behalves.

ZigBee was not able to easily plug its kind of networks into the IP-based Internet [52], and with the increasing trend towards the *Internet-of-Things*, the ZigBee alliance designed the *ZigBee IPv6-based stack* [53] for 802.15.4 networks. This changed the "traditional" ZigBee stack to use 6LoWPAN, to use RPL as routing protocol, and to use UDP.

2.1.5 Hardware Platforms for WSN

WSNs can be deployed in a range of hardware products including *Tmote Sky* (also known as *TelosB*) [54, 18], WiSMote [55], Zolertia Z1 mote [56], MICAz [57], eZ430-RF2500 [58], CC1110 & CC2510 Development Kit [59], and other platforms [60, 61].

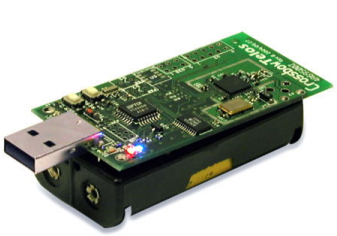

Figure 2.11: Crossbow TelosB Mote

The *Tmote Sky* is an open-source platform developed by University of California, Berkeley [54]. It provides a very power efficient sensor mote, equipped with the Texas Instruments MSP430 micro-processor, the IEEE 802.15.4 compliant Texas Instruments CC2420 RF chip, 8 ADC channels for sensing data, sensors for temperature, light and humidity, as well as several interface ports. USB support is also provided, for an easier programming environment or communication with a PC. The *Tmote Sky* is an architecture that is widely supported for both *TinyOS* and *ContikiOS* applications, as well as the *COOJA* simulator.

There are other platforms available, however we have found that the *TMote Sky* platform suits the needs of our work. The *Crossbow TelosB* [18] shown in Fig. 2.11 is one of the *TMote Sky* architecture products available on the market, and when comparing it with others, we consider it as the most resource limited. We assumed that if we can run our software on it, we will be able to deploy it on any other platform. For these reasons, we have selected it to carry on the tests on this

Table 2.1: TelosB technical specifications

Specifications		Remarks
Module		
Processor Performance	16 bit RISC	
Program Flash Memory	48K bytes	
Measurement Serial Flash	1024K bytes	
RAM	10K bytes	
Configuration EEPROM	16K bytes	
Serial Communications	UART	0-3V transmission levels
Analog to Digital Converter	12 bit ADC	8 channels, 0-3V input
Digital to Analog Converter	12 bit DAC	2 ports
Other Interfaces	Digital I/O,I2C,SPI	
RF Transceiver		
Frequency band	2400 MHz to 2483.5 MHz	ISM band
Transmit (TX) data rate	250 kbps	
Transmit symbol rate)	(62.5 ksymbol/s	
RF power	-24 dBm to 0 dBm	
Receive Sensitivity	-90 dBm (min), -94 dBm (typ)	
Adjacent channel rejection	47 dB	+ 5 MHz channel spacing
	38 dB	- 5 MHz channel spacing
Outdoor Range	75 m to 100 m	Inverted-F antenna
Indoor Range	20 m to 30 m	Inverted-F antenna
Sensors		
Visible Light Sensor Range	320 nm to 730 nm	Hamamatsu S1087
Visible to IR Sensor Range	320 nm to 1100nm	Hamamatsu S1087-01
Humidity Sensor Range	0-100% RH	Sensirion SHT11
Resolution	0.03% RH	
Accuracy	$\pm$3.5% RH	Absolute RH
Temperature Sensor Range	-40^oC to 123.8^oC	Sensirion SHT11
Resolution	0.01^oC	
Accuracy	$\pm$0.5^oC	@25^oC
Electromechanical		
Battery	2X AA batteries	Attached pack
User Interface	USB	v1.1 or higher
Size (in)	2.55 x 1.24 x 0.24	Excluding battery pack
(mm)	65 x 31 x 6	Excluding battery pack
Weight (oz)	0.8	Excluding batteries
(grams)	23	Excluding batteries

thesis. Table 2.1 states the technical specifications of the platform, and Table 2.2 details its power consumption.

Table 2.2: TelosB power consumption

	Nominal
Current in Transmit (0 dBm) mode (mA)	19.5
Current in Receive mode (mA)	21.8
Current in MCU on, radio off (mA)	1.8
Current in MCU on, radio on - idle mode (μA)	365
Current in Sleep mode (μA)	5.1
Power supply (V)	3.6

2.1.6 Operating Systems and Simulation Environments for WSN

Operating System (OS) support for WSNs plays a central role in building scalable distributed applications that are efficient and reliable. Over the years, we have seen a variety of Operating Sistems (OSes) emerging in the research community to facilitate developing WSN applications.

There are some OSes for WSN, and we have selected two of them, TinyOS/TOSSIM and *ContikiOS/COOJA* as we consider them as the most used OSes and simulation environments in the literature. In [61] Wei Dong et al. provide an overview of existing work in this area, present a taxonomy of state-of-the-art OSes, discuss various approaches to address design challenges, and discuss evaluations of a *sensornet* OS and present some recommendations from the perspectives of OS developers and OS users.

2.1.6.1 TinyOS

TinyOS [62] is a free and open-source operating system developed for embedded systems with memory-constrained devices. TinyOS is perhaps the earliest OS for WSN in the literature [61, 63], and is based on an event-driven architecture and is implemented in NesC [64], which is a programming language based in C, thus, limiting the portability of the operating system. Fig. 2.12 shows the *TinyOS* architecture, and in [65] a more in depth description of this architecture is presented and discussed. b6lowpan [66] (usually called blip) is one implementation of 6LoWPAN for TinyOS. Blip is more than a 6LoWPAN implementation based on [67]. It is an IPv6 stack including Neighbor Discovery, support for TCP, UDP, DHCPv6, and has a point-to-point daemon to communicate with Unix machines and is the basis for the TinyRPL [68] and CoAP implementations. To enable a flexible architecture and a low resource consumption, TinyOS programming is based on components which are wired together to create an application at design-time. Component interactions happen at two directions, i.e., one component can use commands provided by another component; also, one component can signal events to another component. The execution model of TinyOS consists of interrupts and tasks. Interrupts execute at a higher priority and can preempt the execution of tasks. Tasks execute at a lower priority and are scheduled in a FIFO manner. Tasks in TinyOS are written in a run-to-completion manner, and they can not be preempted or

self-suspended. For this reason, I/Os are done in split-phases, i.e., a request is done at the end of a task while signal invokes the start of the next task. TinyOS version 2 (T2) provides telescoping abstraction, which is a hybrid of horizontal decomposition (for the lower level to support different kinds of hardware devices) and vertical decomposition (for the higher level to support platform-independent functionality), and makes it easier to support new hardware platforms. Moreover, T2 provides service distribution, which limits arbitrary component compositions and provides a group of components (i.e., services) to improve system reliability. Besides architectural improvements, there are a number of important improvements in implementation, including threading support [61].

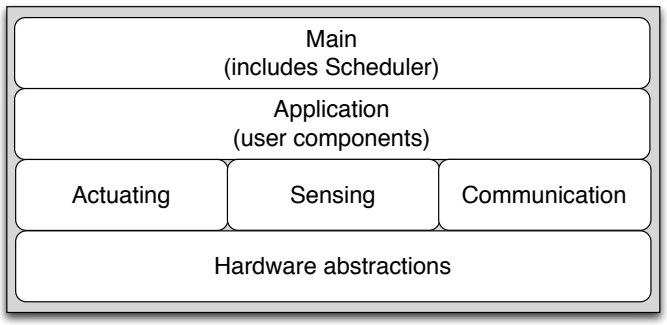

Figure 2.12: TinyOS architecture

TinyRPL [68] is a prototype implementation of the IETF RPL routing protocol in TinyOS 2.x and supports the RPL draft's basic mechanisms, while omitting some of RPL's optional features, such as the security options. The current implementation provides the *OF Zero* (*OF0*) objective functions [42] or the *Minimum Rank with Hysteresis OF* (*MRHOF*) [41] objective function with the ETX metric. Nevertheless, the modular design of *TinyRPL* simplifies the use of additional objective functions.

2.1.6.2 TOSSIM

TOSSIM [69] captures the behavior and interactions of networks of thousands of TinyOS motes at network bit granularity. The TOSSIM architecture is composed of five parts: support for compiling TinyOS component graphs into the simulation infrastructure, a discrete event queue, a small number of re-implemented TinyOS hardware abstraction components, mechanisms for extensible radio and ADC models, and communication services for external programs to interact with a simulation.

TOSSIM takes advantage of TinyOS's structure and whole system compilation to generate discrete-event simulations directly from TinyOS component graphs. It runs the same code that runs on sensor network hardware. By replacing a few low-level components, TOSSIM translates hardware interrupts into discrete simulator events; the simulator event queue delivers the interrupts that drive the execution of a TinyOS application. The remainder of TinyOS code runs unchanged.

TOSSIM uses a very simple but powerful abstraction for its wireless network. The network is a directed graph, in which each vertex is a node, and each edge has a bit error probability. Each node has a private piece of state representing what it hears on the radio channel. This abstraction allows testing under perfect transmission conditions (bit error ratio is zero), can capture the hidden terminal problem (for nodes a,b,c, there are edges (a, b) and (b, c) but no edge (a, c)), and can capture many of the different problems that can occur in packet transmission (start symbol detection failure, data corruption, etc.).

The simulator engine provides a set of communication services for interacting with external applications. These services allow programs to connect to TOSSIM over a TCP socket to monitor or actuate a running simulation. Details of the ADC and radio models, such as readings and loss rates, can be both queried and set. Programs can also receive higher level information, such as packet transmissions and receptions or application-level events.

TOSSIM supports the TinyOS tool-chain, making the transitions between simulated and real networks easy. Compiling to native code allows developers to use traditional tools such as debuggers in TOSSIM. As it is a discrete event simulation, users can set debugger breakpoints and step through what is normally real-time code (such as packet reception) without disrupting operation. It also provides mechanisms for other programs to interact and monitor a running simulation; by keeping monitoring and interaction external to TOSSIM, the core simulator engine remains very simple and efficient.

2.1.6.3 ContikiOS

The *ContikiOS* [70] is an open source operating system for networked embedded systems in general, and wireless sensor nodes in particular. It is developed by a team of developers from the industry and academia.

A running *ContikiOS* system consists of the kernel, libraries, the program loader, and a set of processes. Communication between processes always goes through the kernel, which does not provide a hardware abstraction layer, but lets device drivers and applications communicate directly with the hardware. A process is defined by an event handler function and an optional poll handler function. The process state is held in the process private memory and the kernel only keeps a pointer to the process state. All processes share the same address space and do not run in different protection domains. Interprocess communication is done by posting events [65].

Looking *ContikiOS* from a higher perspective, the system is partitioned into two parts: the core and the loaded programs as shown in Fig. 2.13. Typically, the core consists of the *ContikiOS* kernel, the program loader, the most commonly used parts of the language run-time and support libraries, and a communication stack with device drivers for the communication hardware.

The core is compiled into a single binary image and is usually not modified after deployment. Programs are loaded into the system by the program loader. The program loader is used to load/unload the programs into the system either by using the communication stack or directly to attached storage (as example, the EEPROM).

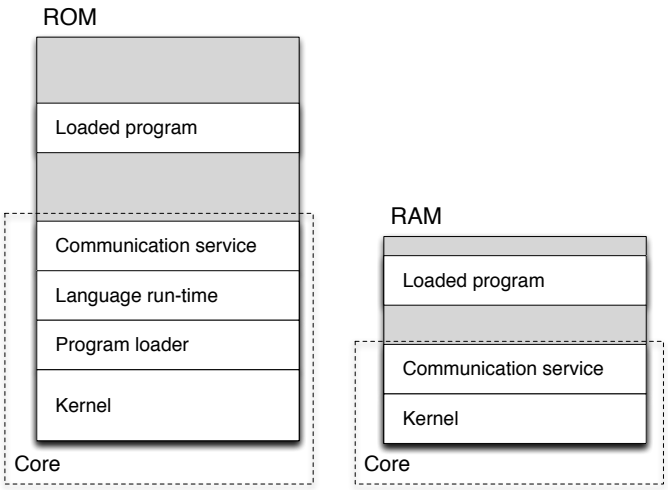

Figure 2.13: ContikiOS system

The *ContikiOS* kernel consists of a lightweight event scheduler that dispatches events to running processes and periodically calls processes' polling handlers. All program execution is triggered either by events dispatched by the kernel or through the polling mechanism. The kernel does not preempt an event handler once it has been scheduled. The kernel supports two kinds of events: asynchronous and synchronous events. In addition to the events, the kernel provides a polling mechanism. Polling can be seen as high priority events that are scheduled in-between each asynchronous event.

Both the *ContikiOS* and applications are implemented in the C programming language. Therefore, *ContikiOS* is highly portable, being ported to a number of microcontroller architectures, in which are included the *Texas Instruments MSP430* and the *Atmel AVR* microcontrollers.

ContikiOS incorporates a implementation of the *IPv6* protocol stack, called *µIPv6* (see figures 2.14 and 2.15), which is optimized for *LLN* devices. The *6LoWPAN* module (known as SICSlowpan project [71]) performs header compression and packet fragmentation [67] before transferring packets to lower layers for transmission. *ContikiOS* provides two *MAC* layers: 1) a Carrier Sensing Multiple Access - Collision Avoidance (*CSMA/CA*) mechanism; and 2) a *NullMAC* mechanism that does not do any *MAC-level* processing. The default *MAC driver* is *CSMA/CA*, which schedules a packet retransmission (up to five times) if the *Radio Duty Clycling* (*RDC*) layer or the radio layer detects a radio collision. The *RDC* layer sits below the *MAC* layer and it implements *RDC* mechanisms to switch ON and OFF the radio transceiver to save energy. *ContikiOS* has several *RDC* drivers, being *ContikiMAC* the default one. *ContikiMAC* [72] is an asynchronous duty cycling algorithm, which uses periodical wake-ups to listen for packet transmissions from neighbors. Specifically, the receivers periodically wake up to check the radio medium for activity using low-power *clear channel assessment* (*CCA*) mechanisms [20].

ContikiOS contains a prototype implementation of the *RPL* standard, called *ContikiRPL* [73]. *ContikiRPL* is the first real-world implementation of RPL developed under *ContikiOS*. In fact, the *ContikiOS*'s academy have developed a comprehensive framework for simulation, experimen-

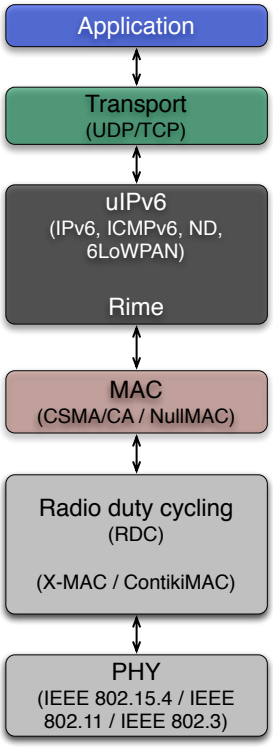

Figure 2.14: ContikiOS protocol stack

tation, and evaluation of RPL routing mechanisms under *ContikiOS*. One of the main features of *ContikiRPL* is that it provides a simple programming interface for designing and evaluating Objective Functions. It consists of four main components [20], shown in Fig. 2.15: 1) a module for message construction and parsing; 2) a module for handling timers; 3) a module that implements rules for route discovery and maintenance; and 4) a module that implements the *Objective Functions* (*OF*). The information exchanged between *ContikiRPL* and other protocol layers occur through callback functions. As example, *ContikiRPL* receives updates on link qualities from μIPv6. Each time *ContikiRPL* receives such updates, it checks if a better parent existing among the node's neighbors according to the chosen *OF* rule. Currently there are two *OFs* supported: 1) the *OF Zero* (*OF0*) [42]; and 2) the *Minimum Rank with Hysteresis OF* (*MRHOF*) [41].

With respect to operating systems for WSN and current RPL implementations, Vu Chien et al. in [65] examine some existing operating systems for WSNs, including *ContikiOS* and *TinyOS*, and compare them by examining some important OS features; *ContikiRPL* implementation and its performance is evaluated in [74], and in [20] where also tradeoffs and inefficiencies are discussed. A detailed analysis of *ContikiRPL* interoperability with, for example, *TinyRPL* can be found in [75] and in [76].

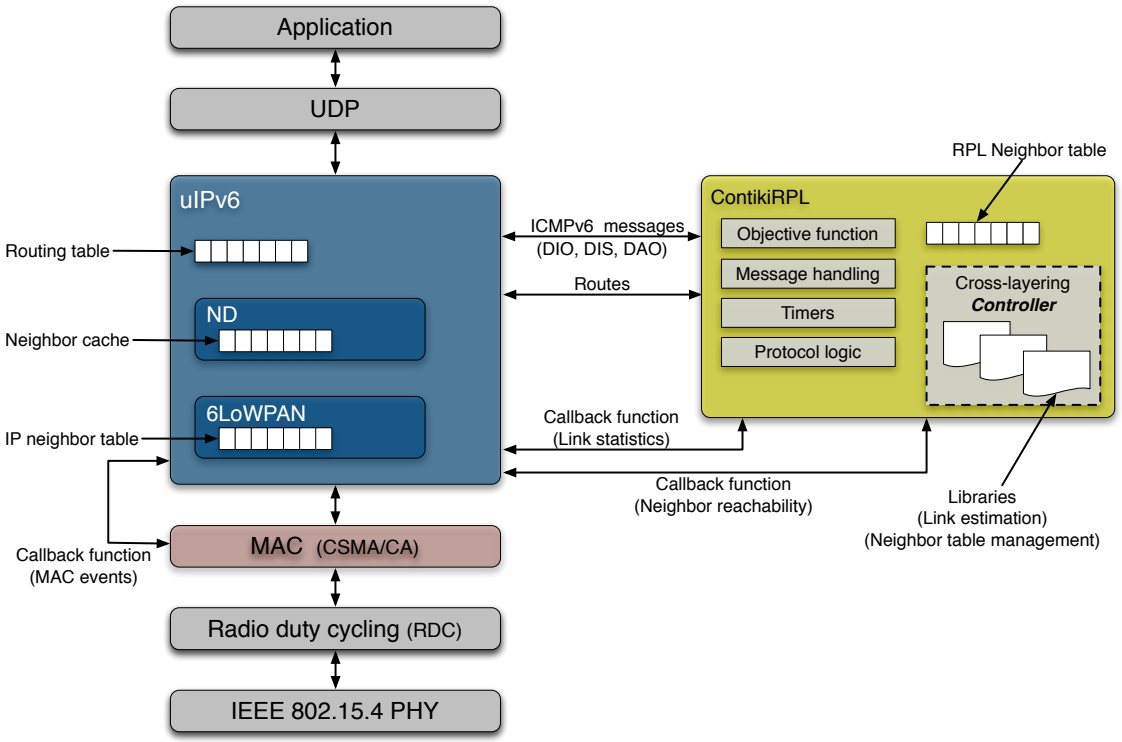

Figure 2.15: ContikiRPL software architecture and protocol stack

2.1.6.4 COOJA

COOJA is a flexible Java-based simulator designed for simulating networks of sensors running the *ContikiOS* [17, 77]. *COOJA* simulates networks of sensor nodes where each node can be of a different type, differing not only in on-board software, but also in the simulated hardware. *COOJA* is flexible in that many parts of the simulator can be easily replaced or extended with additional functionality. Example of parts that can be extended include the simulated radio medium, simulated node hardware, and plugins for simulated input/output.

A simulated node in *COOJA* has three basic properties: its data memory, the node type, and its hardware peripherals. The node type may be shared between several nodes and determines properties common to all these nodes. For example, nodes of the same type run the same program code on the same simulated hardware peripherals. And nodes of the same type are initialized with the same data memory. During execution, however, nodes' data memories will eventually differ due to e.g. different external inputs.

COOJA currently is able to execute *ContikiOS* programs in two different ways. Either by running the program code as compiled native code directly on the host CPU or, for instance, by running compiled program code in an instruction-level TI MSP430 emulator. *COOJA* is also able to simulate non-*ContikiOS* nodes, such as nodes implemented in Java or even nodes running another operating system. All different approaches have advantages as well as disadvantages. Java-based nodes enable much faster simulations but do not run deployable code. Hence, they are useful for the development of e.g. distributed algorithms. Emulating nodes provides more fine-

grained execution details compared to Java-based nodes or nodes running native code. Finally, native code simulations are more efficient than node emulations and still simulate deployable code. Since the need of abstraction in a heterogeneous simulated network may differ between the different simulated nodes, there are advantages in combining several different abstraction level in one simulation. For example, in a large simulated network a few nodes may be simulated at the hardware level while the rest are implemented at the pure Java level. Using this approach combines the advantages of the different levels. The simulation is faster than when emulating all nodes, but at the same time enables a user to receive fine-grained execution details from the few emulated nodes.

COOJA executes native code by making *Java Native Interface* (JNI) calls from the Java environment to a compiled *ContikiOS* system. The *ContikiOS* system consists of the entire *ContikiOS* core, pre-selected user processes, and a set of special simulation glue drivers. This makes it possible to deploy and simulate the same code without any modifications, minimizing the delay between simulation and deployment.

The Java simulator has full control over the memory of simulated nodes. Hence the simulator may at all times view or change *ContikiOS* process variables, enabling very dynamic interaction possibilities from the simulator. The hardware peripherals of simulated nodes are called interfaces, and enable the Java simulator to detect and trigger events such as incoming radio traffic.

All interactions with simulations and simulated nodes are performed via plugins. An example of a plugin is a simulation control that enables a user to start or pause a simulation. Both interfaces and plugins can easily be added to the simulator, enabling users to quickly add custom functionality for specific simulations.

COOJA allows for simultaneous simulations at three different levels, namely the networking (or application) level, the operating system level and the machine code instruction level:

- **Networking Level**: During design and implementation of, for instance, routing protocols, the specific hardware is often not as important as the networking itself. The most important factors may instead concern the radio medium, radio devices and perhaps sleep duty cycles of the sensor nodes. When performing such a design and implementation task it may be possible, but not necessary, to use a fine-grained simulation environment such as an instruction-level simulator. *COOJA* supports code development by enabling the user to easily exchange certain simulator modules such as device drivers or radio medium modules. A simulation can be saved and reloaded using other more or less detailed modules, still with the other simulation parameters unaltered. Furthermore, new radio media and interfaces such as radio devices can easily be developed in Java and be added to the *COOJA* simulation environment.

  To further simplify and speed up development in such scenarios, *COOJA* also supports adding pure Java code nodes. Without any connection to *ContikiOS*, these can be useful when developing high-level algorithms which, when tested and evaluated, will be ported to deployable sensor node code. Pure Java nodes can also be used in heterogeneous networks

where the user only needs to focus on a subset of all the simulated nodes. Since such Java nodes require less memory and processing power, larger heterogeneous network can be simulated more efficiently. For example, using Java nodes, users may rapidly implement the functionality of several different nodes together forming a network. And then later users can, node by node, port the Java code to deployable *ContikiOS* node code, while still maintaining full functionality of the network.

- **Operating System Level**: *COOJA* simulates the operating system by executing native operating system code as described in the previous section. As the entire *ContikiOS*, including any user processes, is executed it is also possible to alter *ContikiOS* core functionality. This is useful for example to test and evaluate changes in included *ContikiOS* libraries.

- **Machine Code Instruction Level**: Using *COOJA*, it is possible to create new nodes with a very different underlying structure than the typical nodes. In [17] it was evaluated this statement by adding nodes connected to a Java-based microcontroller emulator instead of a compiled *ContikiOS* system. The emulated nodes are controlled in a way similar to the native code nodes. Each simulated node is allowed to run for maximum fixed period of time or long enough to handle one event. Events are then, by using the current node memory, transferred via the hardware interfaces to and from the simulator.

- ***COOJA*'s Support for Cross-level Simulation**: *COOJA* supports simulations at these three different abstraction levels. Note that the individual node is always simulated at one of these levels. The main advantage of *COOJA*'s cross-level simulations is that nodes from each of the levels can co-exist and interact in the same simulation. Thus, for example, an emulated node can send a radio packet to a Java based node.

Each simulation in *COOJA* uses a radio model that characterizes radio wave propagation. New radio models may be added to the simulation environment. The radio model is chosen when a simulation is created. This enables a user to, for instance, develop a network protocol using a simple radio model, and then testing it using a more realistic model, or even a custom made model to test the protocol in very specific network conditions. Often a radio model provides one or several plugins in order to configure and view the current simulated network conditions. *COOJA* supports, except from a completely silent model, a simple model that uses an interference and a transmission range parameter that can be changed during a simulation run. Ongoing work on better radio models will provide *COOJA* with a general ray-tracing based model supporting radio absorbing material.

2.1.7 Discussion

This section provided an overview on the technologies and on the protocols we used to achieve the goals of this thesis. We have described the concept of LLN and presented the IEEE 802.15.4 standard. We also described the 6LoWPAN adaptation layer aimed at enabling the transport of IPv6 packets over IEEE 802.15.4.

We have also made an overview of routing in WSN supporting IPv6, namely the 6LoWPAN mesh routing, and RPL routing protocol. With respect to RPL, we have made an deep characterization of the protocol and described and discussed existing implementations such as *TinyRPL* and *ContikiRPL*.

This section also provided an overview of the available hardware platforms for wireless sensor nodes, and provided an overview of the available operating systems and simulators for WSNs with a special emphasis on the *ContikiOS* and on the *COOJA* simulator. Besides the operating systems described, there are others such *LiteOS* [78] and RIOT OS [79, 80, 81].

Finally, we have selected *ContikiOS* and *COOJA* to perform the simulation and experimentation throughout this thesis. Our choice was due to the ease in programing and to its ease portability, because the typical *ContikiOS* memory footprint using full μIPv6 networking and RPL requires only 10 kBytes of RAM and 30 kBytes of ROM, which represents 100% of available RAM and 62.5% of available ROM of the *Tmote Sky* sensor node [82], and because it supports the RPL routing protocol that has been established as the *"de-facto"* standard routing protocol for WSN.

2.2 Energy Efficiency

Energy efficiency is about energy conservation, i.e., how to reduce the energy consumption of the nodes evolving in all the nodes activities, including sensing, processing and communication tasks, so that the network lifetime can be extended to reasonable times. In the following we discuss in detail some energy related aspects including main sources of WSN nodes power consumption, energy consumption in WSN and energy conservation in WSN.

2.2.1 Main Sources of WSN Nodes Power Consumption

Sensor nodes are usually battery powered and deployed in large areas in which changing or replacing batteries may be impractical or even unfeasible. Therefore, minimizing the power consumption in a node is a primary issue to be considered, and the use of solutions for increasing the nodes lifetime is fundamental in WSN. It is well known that energy consumed by nodes in data sensing or processing functions may be negligible [83]. In contrast, data communication has a strong impact on the nodes battery mainly because of two aspects: 1) the radio transceiver implies a high power consumption when compared to the other components of the node; and 2), the communications phase is associated with phenomena such as collisions, overhearing, overemitting, and idle listening, which substantially reduce the nodes battery [84]. Furthermore, in a WSN, each node plays the dual role of data originator and data router. In the sensor's data originator role, each sensor gathers data from the environment through various sensors nodes which need to be processed and transmitted to nearby sensor nodes for multi-hop delivery to the sink. On the other hand, in the sensor's data router role, in addition to originating data, each sensor node is responsible for relaying the information transmitted by its neighbors, whereas the low-power communication techniques in WSNs limit the communication range of a sensor node. In a large network, multi-hop communication is required so that sensor nodes relay the information

sent by their neighbors to the sink node. Accordingly, the sensor node is responsible for receiving the data sent by its neighbors and forwarding it to one of its neighbors according to the routing decisions [85].

Pantazis et al. [83] present a survey on energy consumption based on the hardware of a typical sensor node. They divide the sensor node into four main components: a sensing subsystem including one or more sensors for data acquisition, a processing subsystem including a microcontroller and memory for local data processing, a radio subsystem for wireless data communication, and a power supply unit. Fig. 2.17 shows the WSN node architecture proposed by them, constituted by a *Sensor Module*, by a *Processing Module*, by *Wireless Communication Module* and by a *Power Supply Module*. These modules work together in order to make the sensor

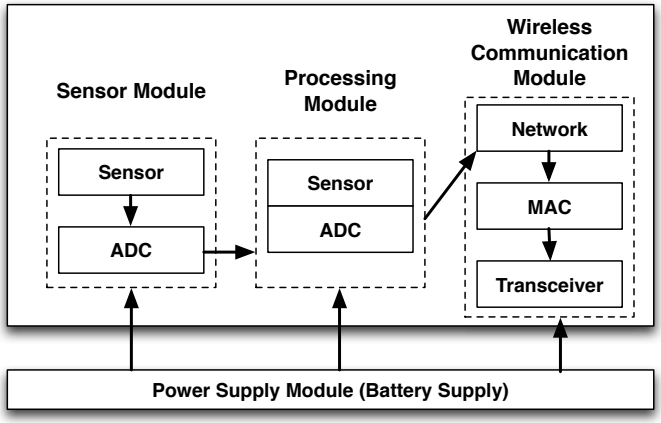

Figure 2.16: The architecture of a WSN node

nodes operational in a WSN environment. Therefore, in order to evaluate the energy consumption of the sensor nodes, it is important to study the energy consumption of their components. There are a few attempts to propose and discus about models for energy efficiency in WSNs. Most of them consider the sensor node power consumption model and, at same time, the impact of the sensor node hardware and external radio environment.

The energy consumption of the wireless sensor nodes based on Fig. 2.16 depends on its components and is summarized on the following:

- **Sensor Module**: the energy consumption of sensor module is due to signal sampling, ADC (Analogue to Digital) signal conversion and signal modulation. Also the energy consumption of this module is related to the sense operation of the node (periodic, sleep/wake, etc.). For example in periodic mode the energy consumption is modeled as [83]

$$E_{sensor} = E_{on-off} + E_{off-on} + E_{sensor-run} \tag{2.1}$$

  where E_{on-off} is the one time energy consumption of closing sensor operation, E_{off-on} is the one time energy consumption of opening sensor operation and $E_{sensor-run}$ is the energy consumption during the time interval of sensing operation.

- **Processing Module**: sensor controlling, protocol communication and data processing are the main activities of this module. This module supports mostly three operation states (sleep, idle and run). The Processor energy consumption, denoted as E_{CPU} is the sum of the state energy consumption $E_{CPU-state}$, and the state-transition energy consumption $E_{CPU-change}$ state, modeled as [83]

$$\begin{aligned} E_{CPU} &= E_{CPU-state} + E_{CPU-change} = \\ &\sum_{i=1}^{m} P_{CPU-state}(i) \cdot T_{CPU-state}(i) + \\ &\sum_{j=1}^{n} N_{CPU-change}(j) \cdot e_{CPU-change}(j) \end{aligned} \tag{2.2}$$

  where $P_{CPU-state}(i)$ is the power of state i that can be found on the node's reference manual, $T_{CPU-state}(i)$ is the time interval in state i, $N_{CPU-change}(j)$ is the frequency of state transition j and $e_{CPU-change}(j)$ is the energy consumption of one-time state transition J.

- **Wireless Communication Module**: the total power consumption for transmitting (PT) and for receiving (PR), is denoted as [83]

$$\begin{aligned} P_T(d) &= P_{TB} + P_{TRF} + P_A(d) \\ &= P_{T0} + P_A(d) \end{aligned} \tag{2.3}$$

  where

$$\begin{aligned} P_R &= P_{RB} + P_{RRF} + P_L \\ &= P_{R0} \end{aligned} \tag{2.4}$$

  where $P_A(d)$ is the power consumption of the power amplifier which is a function of the transmission range d, and it depends on many factors including the specific hardware implementation, DC bias condition, load characteristics, operating frequency and P_A output power.

- **Power Supply Module**: this module is related to the manufacturer and the model of each node. For example, a wireless sensor node *TelosB* is supplied by two AA batteries, while the current draw on receive mode is 21mA and on transmit for Tx value 0dBm is 19.5mA. Also on the sleep mode it consumes less than 5.1 *mu*A.

G. Anastasi et al, in their work [86], also present a sensor node-level architecture, as shown in Fig. 2.17, which is usually assumed in the literature. It consists of four main components: (1) a sensing subsystem including one or more sensors (with associated analog-to-digital converters) for data acquisition; (2) a processing subsystem including a micro-controller and memory for local data processing; (3) a radio subsystem for wireless data communication; and (4) a power supply unit. Depending on the specific application, sensor nodes may also include additional components such as a location finding system to determine their position, a mobilizer to change their location or configuration (e.g., antenna's orientation), and so on. Obviously, the power breakdown heavily depends on the specific node, and the following remarks generally hold [87]:

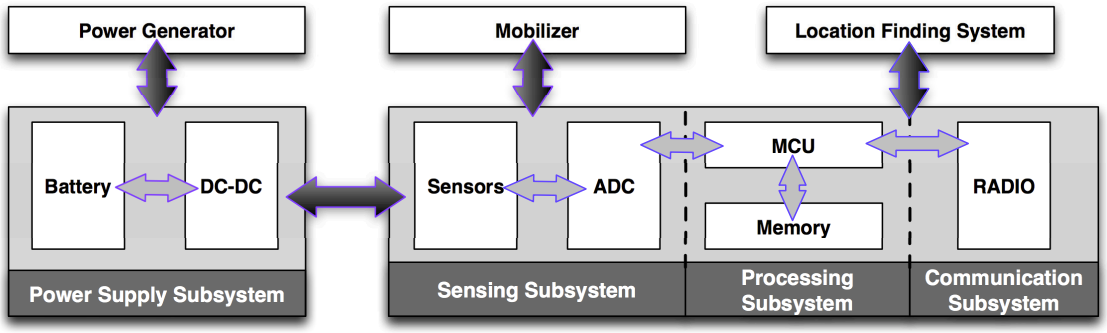

Figure 2.17: Anastasis's architecture of a typical WSN node

- The communication subsystem has an energy consumption much higher than the computation subsystem. G. Pottie [88] shown that transmitting one bit may consume as much as executing a few thousands instructions. Therefore, communication should be traded for computation.
- The radio energy consumption is of the same order of magnitude in the reception, transmission, and idle states, while the power consumption drops of at least one order of magnitude in the sleep state. Therefore, the radio should be turned off whenever possible.
- Depending on the specific application, the sensing subsystem might be another significant source of energy consumption, so its power consumption has to be reduced as well.

Dunkels et al [89], about this same subject, propose Eq. 2.5 which models the energy consumption based on the hardware components of a typical sensor node, E (J), as

$$\begin{aligned} E &= (I_m \times t_m + I_l \times t_l + I_t \times t_t + \\ &\quad I_r \times t_r + \sum_{i=1}^{n} I_{c_i} \times t_{c_i}) \times V \end{aligned} \tag{2.5}$$

where I_m is the current consumed by the microprocessor in the time t_m during which the microprocessor is running, I_l and t_l are the current and time when the microprocessor is in low power mode, I_t and t_t are the current and the time when the device communication is in transmit mode, I_r and t_r are the current and the time when the device communication is in receive mode, I_{c_i} and t_{c_i} are the current and the time consumed by other components (e.g. LEDs, ADCs, DACs), and V the sensor supply voltage. This equation is used in some operating systems such as *ContikiOS* [16] to estimate the energy consumption of a node when it reduces the energy consumption by powering off the microcontroller or other hardware components when they are not used.

An up to date approach regarding the energy consumption of the WSN nodes is presented in [90].

2.2.2 Energy Consumption in WSN

Some discussion and models on energy efficiency in WSN can be found on the literature, where technical approaches for prolonging the lifetime of battery-powered sensors have been the

focus. These approaches include energy-aware protocol development and hardware optimizations, such as power consumption models for WSN devices and sleeping schedules to keep electronics inactive most of the time. Pantazis et al [83] provide a good overview on this topic. They discuss a number of developed energy efficient routing protocols and models, and provide directions to select the most appropriate to be used in different WSN applications. According to them, most of the WSNs energy consumption is spent on three main activities: sensing, data processing and communication. All of these are important factors that should be considered when developing new protocols for WSNs. Communication in WSN is the major component of the energy consumption. Therefore, current research in WSNs is mostly concentrated on designing routing protocols that use the less possible energy during the communication of the nodes. The main objective of these protocols is not only to find the lowest energy path from a source to a destination, but also to find the most efficient way to extend the network's lifetime.

Pantazis et al, in [83], looked at L. Alazzawi et al work [91] and defined some terms related to WSNs energy efficiency. These terms are used to evaluate the performance of the routing protocols and bellow we describe the most important related to our work:

- **Energy per Packet**: refers to the amount of the energy that is spent while sending a packet from a source to a destination.

- **Network Lifetime**: there is no unique definition for the network lifetime. In many cases the term network lifetime corresponds to the time when the first node exhausts its energy, or when a certain fraction of the network's nodes is dead, or even when all nodes are dead. However, the importance of a WSN is to be operational and able to perform its tasks during its use. In WSNs, it is important to maximize the network lifetime, which means to increase the network survivability or to prolong the battery lifetime of nodes. Moreover, the lifetime of a node is effectively determined by its battery life. The main drainage of battery is due to transmitting and receiving data among nodes and the processing elements.

- **Average Energy Dissipated**: is a metric related to the network lifetime and shows the average dissipation of energy per node over time in the network as it performs various functions such as transmitting, receiving, sensing and aggregation of data.

- **Low Energy Consumption**: a low energy protocol has to consume less energy than traditional protocols. This means that a protocol that takes into consideration the remaining energy level of the nodes and selects routes that maximize the network's lifetime is considered as low energy protocol.

- **Average Packet Delay**: is a metric to compute as the average one-way latency that is observed between the transmission and reception of a data packet at the sink. This metric measures the temporal accuracy of a packet.

- **Time Until The First Node Dies**: is a metric that indicates the duration for which all the sensor nodes on the network are alive. There are protocols in which the first node on the

network runs out of energy earlier than in other protocols, but manages to keep the network operational much longer.

- **Idle Listening**: a sensor node that is in *idle listening mode*, does not send or receive data, it can still consume a substantial amount of energy. Therefore, this node should not stay in idle listening mode, but should be powered off.

- **Packet Size**: the size of a packet determines the time that a transmission will last. Therefore, it is effective in energy consumption. The packet size has to be reduced by combining several packets into one large packet or by compression.

- **Distance**: the distance between the transmitter and the receiver can affect the power that is required to send and receive packets. The routing protocols can select the shortest paths between nodes and reduce energy consumption.

2.2.3 Energy Conservation in WSN

The failure of a few nodes can cause topological changes and might require rerouting of data packets and reorganization of the network. In this regard, power conservation and management take on additional significance.

In [86], G. Anastasi et al. describe a generic WSN node architecture and its associated power breakdown which they use to propose a solution for reducing consumption in WSN. They describe different approaches for energy management and conclude that the energy consumption of the radio is much higher than the energy consumption due to data sampling or data processing. Based on their wireless sensor node architecture and power breakdown, described in Sec. 2.2.1, they have exploited several approaches to reduce the power consumption. At a very general level, they have identified three main enabling techniques, namely, *duty cycling*, *data-driven approaches*, and *mobility* (see Fig. 2.18).

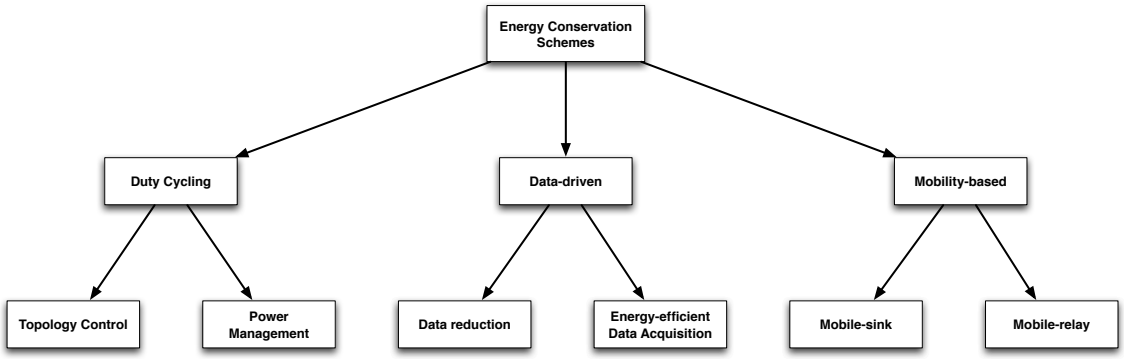

Figure 2.18: Taxonomy of approaches to energy saving in WSN

The *design of duty cycle schemes* is a technique that tries to increase the nodes lifetime by scheduling the nodes radios states depending on network activity. According to G. Anastasi et al.

[86] duty cycling can be achieved through two different and complementary approaches: *power management*, and *topology control*.

In the case of the *power management* approach, it is possible to exploit node redundancy, and adaptively select only a minimum subset of nodes to remain active for maintaining connectivity; nodes that are not currently needed for ensuring connectivity can go to sleep and save energy. The concept of *power management* is associated with *sleep/wakeup* schemes which can be defined for the radio subsystem of the sensor node, without relying on topology or connectivity aspects. *On-demand* protocols are examples. The basic idea is that a node should wakeup only when another node wants to communicate with it. The main problem associated is how to inform the sleeping node that some other node is willing to communicate with it. A possible solution consists in using a *scheduled rendezvous* approach. The basic idea is that each node should wake up at the same time as its neighbors, according to a wakeup schedule, and remain active for a short time interval to communicate with their neighbors. Then, they go sleep until the next rendezvous time. The major advantage of such scheme is that when a node is awake it is guaranteed that all its neighbors are awake as well. This allows sending link-local multicast messages to all neighbors. On the opposite, scheduled rendezvous schemes require nodes to be synchronized in order to wake up at the same time. This scheme, as others such as *On-demand* and *Asynchronous* are surveyed and discussed in [86].

In the case of the *topology control* approach, finding the optimal subset of nodes that guarantee connectivity is the main objective. Therefore, the basic idea behind *topology control* is to exploit the network redundancy to prolong the network longevity. Moreover, active nodes do not need to maintain their radio continuously on. They can switch off the radio when there is no network activity, thus alternating between sleep and wakeup periods. These two approaches are complementary and implement *duty cycling*. The concept of *topology control* is associated with that of network redundancy. Large WSN typically have some degree of redundancy. In many cases network deployment is done at random. Therefore, it may be convenient to deploy a number of nodes greater than necessary to cope with possible node failures occurring during or after the deployment. *Topology control* protocols are thus aimed at dynamically adapting the network topology, based on the application needs, so as to allow network operations while minimizing the number of active nodes. There are some criteria to decide which nodes to activate/deactivate, and when. So, *topology control* protocols are classified in two categories: *location driven* protocols, and *connectivity driven* protocols. *Location driven* protocols define which node to turn on and when, based on the location of sensor nodes which is assumed to be known. *Connectivity driven* protocols, dynamically activate/deactivate sensor nodes so that network connectivity, or complete sensing coverage [92], are fulfilled. A detailed survey on topology control in wireless ad hoc and sensor networks is available in [93].

Other techniques that try to increase the nodes lifetime use *data-driven approaches*, such as the *data aggregation* scheme. This scheme tries to address the case of unneeded samples, aimed at reducing the energy spent by the sensing subsystem. Some of these schemes can also reduce the energy spent for communication as well, as they reduce the amount of data to be delivered to

the sink node. Data aggregation is achieved at intermediate nodes between the sources and the sink to reduce the amount of data traversing the network towards the sink. The most appropriate data aggregation technique depends on the specific application and must be tailored to it. We do not detail this technique because it is not related to our work but, being application-specific, in [86, 94] we can find a comprehensive and up-to-date survey about *Data-driven* approaches.

In case some of the sensor nodes are mobile, *mobility* can be used as a tool for reducing energy consumption. In a static sensor network packets coming from sensor nodes follow a multi-hop path towards the sink(s). Thus, a few paths can be more loaded than others, and nodes closer to the sink have to relay more packets so that they are more subject to premature energy depletion. If some of the sensor nodes (including the sink) are mobile, the traffic flow can be altered if mobile nodes are responsible for data collection directly from static nodes. Static nodes wait for the passage of the mobile node and route messages towards it, so that communication takes place in proximity. As a consequence, static nodes can save energy because path length, contention and forwarding overheads are reduced as well [86]. Nevertheless, mobility is out of scope of this Thesis.

2.2.4 Discussion

This section provided an overview on main sources node power consumption, WSN energy consumption, and WSN energy conservation. The main sources of node power consumption were analyzed based on node's architecture. For the WSN energy consumption we have defined terms related to energy efficiency that are commonly found in the literature; these terms are used to evaluate the performance of the routing protocols designed for WSNs. Finally, in what concerns energy conservation in WSN, we have presented and described a taxonomy of approaches used in the literature to save energy in WSN.

2.3 Time Synchronization

WSNs are constituted by sensor devices equipped with their own local clock for internal operations [95]. Events related to them, which include sensing, processing, and communication, are normally associated to timing information. In the particular case of WSN there are many challenges related to time synchronization because these networks are distributed by nature, and because of the constraints of the sensor nodes in terms of hardware and of software.

Akyildiz et al. [95] state that in order for the nodes synchronize, they must exchange information about their clocks and use this information to synchronize their local clocks. By using wireless communications, WSNs create challenges for synchronization that result from the error-prone communication nature of the wireless channel which may cause packet losses due to low signal to noise plus interference ratios, or highly and variant non-deterministic delays caused by MAC access and packet retransmissions. These factors affect also the time synchronization messages. Therefore, some nodes may be unsynchronized. On the other hand, synchronization messages sent by nodes may lead other nodes to adapt to their unsynchronized local clocks. As a consequence, the network may be partitioned into different areas with different time, that prevents

synchronization of the entire network. Also, the wireless channel may introduce asymmetric delays between two nodes, which is important for synchronization because some synchronization solutions depend on consecutive message exchange and round-trip-time delays. Therefore, robust synchronization methods are needed.

We start by identifying some factors that influence the synchronization of the nodes, and that should be considered in the design of time synchronization mechanisms for WSN. As the *Network Time Protocol* (NTP) protocol [96] is a synchronization protocol normally used in IP networks, we provide an overview of it and also describe synchronization protocols for WSN related to our work.

2.3.1 Factors Influencing Time Synchronization

According to [97], some of the factors influencing time synchronization in large systems constituted for example by personals computers, also apply to sensor networks, where temperature, phase noise, frequency noise, asymmetric delays, clock glitches, and sensors constraints are examples of these factors. In the case of the *temperature*, since sensor nodes are deployed in various places, temperature variations throughout the day may cause the clock to speed up or slow down. In the case of the *Phase noise* factor, some of its causes are due to fluctuations in the hardware interface, response variation of the operating system to interrupts, and jitter in the network. The *frequency noise* results from the instability of the clock crystal. In the *Asymmetric delay* factor, the delay of the path from one node to another node may be different than the return path which may result in an asymmetric delay and may cause an offset to the clock, which may go undetected. *Clock glitches* are abrupt jumps in time, caused by hardware or software anomalies such as frequency and time steps. Finally, WSN nodes are constrained by nature because of limited resources (e.g. low in energy consumption, low in processing power, or low in memory).

The transmission and reception of packets are the factor that causes more energy consumption in a sensor node. Therefore, a time synchronization protocol for sensor networks should help overcoming the synchronization problems introduced by the factors described above, avoid frequent message exchanges and be self-configurable.

2.3.2 Network Time Protocol

The Network Time Protocol (NTP) [96] is the synchronization protocol more often used in the Internet. This protocol includes several synchronization mechanisms that have been also adapted for developed WSN synchronization protocols. RBS [1], TPSN [2], LTS [3] and TSync [4] are some examples of these protocols. NTP is used to adjust the clock of each network node. This synchronization is achieved by using a hierarchical structure of time servers. The root node is synchronized with the *Coordinated Universal Time* (UTC). In each level of this hierarchy, the time servers nodes synchronize the clocks of their subnetwork peers. NTP uses a two-way handshake between two nodes to estimate the delay between these nodes and computes the relative offset accordingly (see Fig. 2.19, where node *s* will synchronize himself with node r). However, NTP

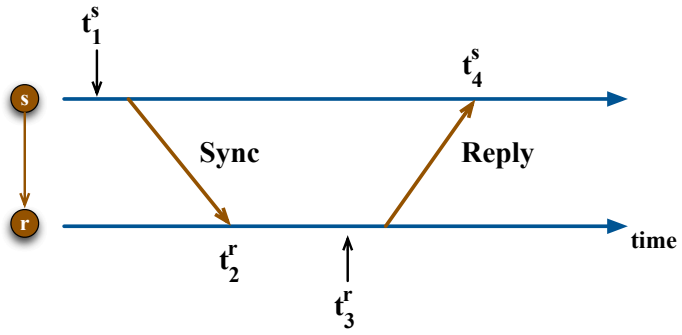

Figure 2.19: NTP two-way handshake mechanism

assumes that the transmission delay between two nodes is the same in both directions. This is reasonable for the Internet, but some of the characteristics of WSN make this assumption inadequate. NTP is useful to discipline the oscillators of the sensor nodes, but using it to connect to time servers may be impossible because of sensor node failures, which are frequent in WSN. Using a single clock reference to synchronize all the nodes could be a problem due to the variations in network delays. Moreover, NTP requires intensive computing, requires a precise time server to synchronize the nodes, and does not consider the energy the nodes may spent to synchronize their clocks. All these problems may cause NTP to inaccurately measure delays and inaccurately estimate clock offsets.

2.3.3 Synchronization Protocols for WSN

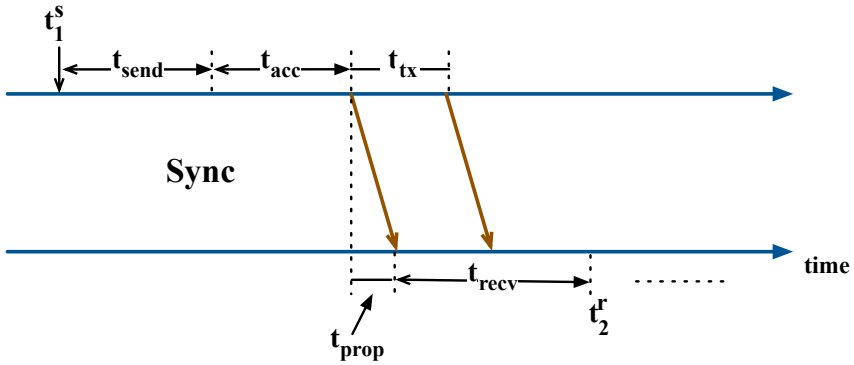

Figure 2.20: Synchronization delay between a pair of nodes

WSN pose unique challenges in the design of synchronization protocols, which calls for specific synchronization solutions. An example is the effect of the broadcast wireless channel. However, wireless communication introduce random delays between two nodes. Let us consider Fig. 2.20, which represents a handshake scheme. The delay between two nodes is characterized by four components: i) the sending delay (t_{send}); ii) the access delay (t_{acc}); iii) the propagation time (t_{prop}); and iv) the receiving delay (t_{recv}).

- **Sending Delay** (t_{send}): corresponds to the handshake between a pair of nodes. The handshake initiates when node s issues a SYNC packet with the timestamp t_1^s. Between the time the synchronization protocol issues the synchronization command and the time during which the SYNC packet is prepared, there is a delay resulting from the combination of operating system delays and transceiver delays on the node's hardware. Moreover, t_{send} is non-deterministic because of the interactions between hardware and software components;
- **Access Delay** (t_{acc}): corresponds to the additional delay introduced by the wireless channel after the packet has been prepared and transferred to the transceiver. This delay depends on the MAC protocol when the node waits for accessing the channel; as an example, MAC protocols using CSMA introduce a significant amount of access delay when the channel is very occupied;
- **Propagation Delay** (t_{prop}): is the amount of time needed to transmit a SYNC packet to a receiver;
- **Receiving Delay** (t_{recv}): is the time required for the transceiver of the receiver node r to receive the packet and process it. The transmission delay, t_{tx}, is a component of the receiving delay, which is important and characterized by the time needed for the SYNC packet to be completely received (see Fig. 2.20); it depends on the transmission rate and on the length of the SYNC packet.

These components contribute to the overall communication delay, also referred as *critical path*. Delays are non-deterministic and create challenges when estimating clock offsets using the NTP's methods. Most of the synchronization protocols for WSN tend to minimize the effects of these delays, which are random. In what follows, 4 related existing synchronization protocols are described, namely RBS [1], TPSN [2], LTS [3] and TSync [4].

2.3.3.1 Reference-Broadcast Synchronization (RBS) [1]

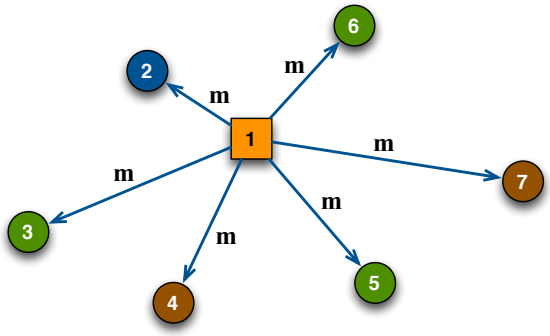

Figure 2.21: Reference Broadcast. Node 1 broadcasts m messages which are used by the other nodes for synchronization purposes

In Fig. 2.21 is shown a *sender-receiver handshake scheme* which introduces a significant amount of non-deterministic delay [1]. The *Reference-Broadcast Synchronization* protocol (RBS)

tries to minimize the overall communication delay in the synchronization process. It eliminates the effect of the broadcast node. Instead of synchronizing the receiver with the sender, RBS synchronizes a set of receivers that are within the reference transmission of a sender. Considering that propagation times are negligible on wireless channels, as soon as a packet is transmitted, it is received at all sender's neighbors almost at the same time. Therefore, the synchronization may be improved if only the receivers are synchronized. As shown in Fig. 2.21, node 1 broadcasts *m* reference packets and each one of the receivers, within its broadcast range, records the time the packets are received. Then, the receiver nodes communicate with each other to estimate the offsets, just like the traditional synchronization. Fig. 2.22 a) shows the *critical path* for traditional synchronization. Sending delays and the access delays should be accurately estimated to improve the synchronization. *Reference Broadcast* synchronization does not involve node 1 in the synchronization; only the receivers (nodes 2, 3, 4, 5, 6 and 7) synchronize among themselves based on a reference broadcast message from node 1. As shown in Fig. 2.22 b), this reduces the *critical path* duration. In fact, the possible origin of uncertainty in RBS is the time between when a broadcast packet is received and when it is completely processed.

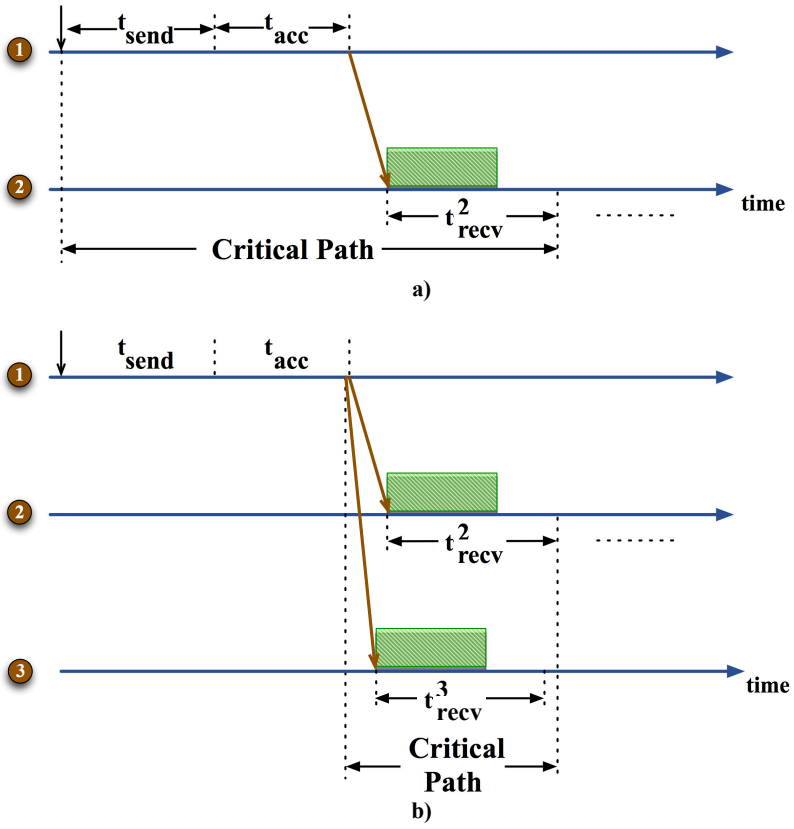

Figure 2.22: Critical path for different synchronization approaches

A method used to determine with efficiency the clock offset of each node in relation to its neighbors is the *receiver-receiver synchronization* method. By exchanging messages with each neighbor, a node fills a table consisting of relative offsets. Therefore, the main goal of RBS is not to correct the clocks of the nodes but, every time a packet is received, to translate its timestamp to the

node's clock using the relative offset information. This synchronization method can only provide synchronization in a broadcast area. In order to provide multi-hop synchronization, RBS uses nodes that receive two or more different reference broadcasts messages. These nodes are called *translation nodes* and they are used to translate the time between different broadcast domains (see Fig. 2.23). As it can be observed, nodes A, B, and C are respectively the transmitter, the receiver, and the translation nodes. The transmitter node broadcasts its timing messages, the receiver node receives those messages and, then, nodes synchronize with each other.

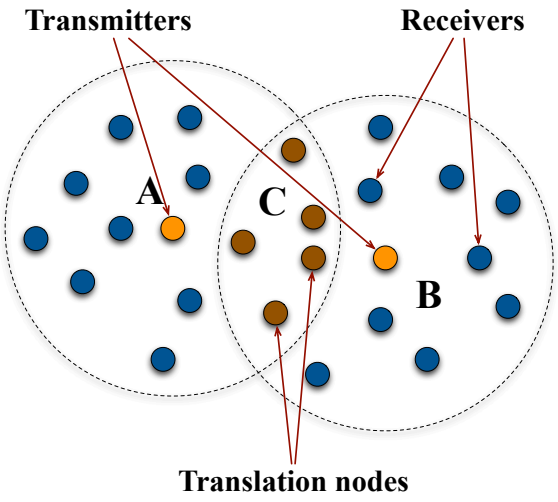

Figure 2.23: RBS multi-hop synchronization scheme

2.3.3.2 Timing-sync Protocol for Sensor Networks (TPSN) [2]

TPSN uses some of the NTP concepts: it uses a hierarchical structure to synchronize the entire WSN to a single time server. TPSN uses the root node to synchronize all or part of the network, consisting of two phases: (1) the *discovery phase*, where the structure of TPSN is built, starting from the root node; and (2) the *synchronization phase*, where pairwise synchronization is performed across the network. In (1) the root node is assigned to level 0, and to the other nodes in the network are assigned to levels according to their distance to the root node (see Fig. 2.24).

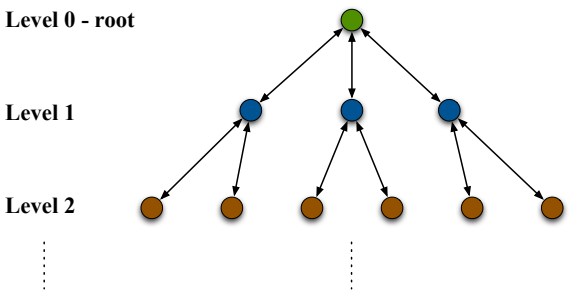

Figure 2.24: Synchronization architecture of TPSN

Firstly, the root node starts to construct the TPSN structure. To this end, it broadcasts a special packet called *level_discovery packet*. In this structure, to the first level is assigned the number 0, which is the level of the root node. The other nodes that receive this packet are the nodes that belong to level 1. Afterward, these nodes broadcast their *level_discovery* packet. Then, the neighbor nodes receiving those packets are labeled as level 2 nodes, and the process is repeated until all the nodes in the network are assigned to a level.

In (2) each node in the structure is synchronized with a node from a higher level. The root node sends another packet (the *time_sync* packet) which initializes the time synchronization process. Afterwards, the nodes in the next level start to synchronize with the root node by sending a *synchronization_pulse* to it, as shown in Fig. 2.25. In order to avoid collisions with other nodes, each node in level 1 waits for a random amount of time before transmitting the *time_sync* packet. After the reception of this packet, the root node sends an acknowledgment back to finish the synchronization process. In this way, nodes belonging to level 1 of the structure are synchronized with root node (see Fig. 2.25). This *time_sync* packet also serves as a *synchronization_pulse* to level 2 nodes. Upon a reception of this packet from a node in level 1, the nodes in level 2 wait for a random amount of time for the level 1 nodes to finish their synchronization. Then, they initialize the synchronization process by transmitting a *synchronization_pulse*. Acting like the root node in level 0, a level 1 node sends back an acknowledgment, the process continues until all the nodes at different levels are synchronized, and the entire network becomes synchronized.

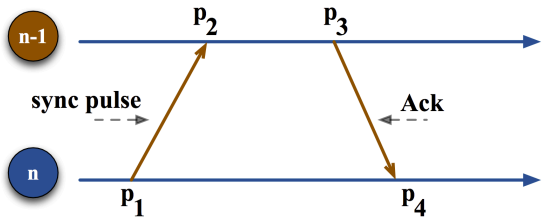

Figure 2.25: Two-way message handshake

In TPSN the receiver synchronizes with the local clock of the sender according to the two-way message handshake, as shown in Fig. 2.25. For this reason, TPSN is based on a *sender-receiver synchronization method*. Hierarchical structures created by TPSN are similar to the structures created by NTP. Like in NTP, nodes may fail causing nodes to become unsynchronized. Also, nodes mobility can make the hierarchy useless, as they may move out of their levels. Therefore, nodes at level n cannot synchronize with nodes at level $n-1$, without requiring additional and periodical synchronization.

2.3.3.3 Lightweight Tree-Based Synchronization (LTS) [3]

LTS is similar to TPSN and follows two design approaches: centralized and distributed. The centralized design is based on the construction of a tree such that each node is synchronized to the root node. After the tree is constructed, the root initiates pairwise synchronization with its children nodes and the synchronization is propagated along the tree to the leaf nodes.

In the distributed design, LTS does not rely on the construction of a tree and synchronization can be initiated by any node in the network. Each node performs synchronization only when it has a packet to send. Therefore, each node is informed about its distance (in number of hops) to the reference node for synchronization, the desired accuracy, the clock drift, and a record of the time that has passed since they were synchronized. Then, the nodes adjust its synchronization rate accordingly. Nodes farther apart from the reference node perform synchronization more frequently because synchronization accuracy is inversely proportional to distance.

In general, LTS is based on message exchanges between two nodes to estimate the clock drift between their clocks. This synchronization scheme is named pairwise synchronization scheme and it is extended for multi-hop synchronization.

2.3.3.4 TSync [4]

TSync combines tree-based synchronization with *receiver-receiver* synchronization method. To this end, two versions of the protocol are developed: (1) *hierarchical referencing time synchronization* and (2) *individual time request*. Version (1) is a centralized protocol where the synchronization is initiated through the root node of the tree. In contrast, version (2) is a distributed protocol where the synchronization mechanism is initiated by any network node. As a consequence, the nodes synchronize between each other by using the *receiver-receiver* synchronization method. Since additional traffic is generated in the network, *TSync* handles the contention and the latency of the synchronization messages by providing support to the MAC layer by reserving a dedicated channel for synchronization messages purposes. Therefore, the synchronization messages are not affected by other data traffic in the network. However, by introducing this enhancement to the MAC layer to improve the synchronization accuracy, it is also increased the complexity required from the nodes because multichannel communication is necessary. Comparing *TSync* with *RBS*, which also uses the *receiver-receiver* synchronization method, the centralized implementation of *TSync* performs better than RBS, while the distributed implementation does not reach the limits of accuracy provided by RBS. Nevertheless, the total number of messages exchanged by the nodes is decreased. Therefore, energy consumption in the synchronization procedures is lower.

2.3.4 Discussion

RBS eliminates the sender-side uncertainty from the critical path. This decreases the synchronization error and improves the efficiency. Each node stores the offset and skew of its neighbors and the time is translated between nodes according to this information. As a result, local clocks are not corrected for each synchronization attempt. Moreover, the time synchronization mechanism is tunable and lightweight since it relies on broadcast messages only. In addition, multi-hop synchronization is provided through gateway nodes, which translate time from one broadcast neighborhood to another. RBS is applicable to any medium which has broadcast capabilities including wired and wireless networks. RBS requires close coupling between the synchronization

structure and the routing protocol. The route from any node to the sink should contain the translation nodes so that the time translation can be performed on the packet along its way to the sink. With the lack of this coupling, there may not be translation nodes on the route along which the message is relayed. As a result, synchronization services may not be available on some routes. RBS is not suitable for TDMA-based MAC protocols since network-wide synchronization is not provided. Compared to the traditional sender-receiver synchronization, e.g., TPSN [2], receiver-receiver synchronization necessitates each node exchanging timing information between its neighbors. RBS requires message exchanges with all the neighbors, which translates into $O(n^2)$ message exchanges if there are n nodes in a node's broadcast range. This increases the energy consumption and may lead to frequent collisions when the network density is high. The large number of message exchanges between neighbors also increases the convergence time of RBS because of possible packet errors and collisions in the broadcast wireless channel. Furthermore, the reference sender cannot be synchronized through RBS. This requires additional synchronization rounds for the senders to be synchronized with the network [95].

The hierarchical structure of TPSN provides scalability. The synchronization of a node depends on its parent in the hierarchical structure. Therefore, even if the number of nodes in the network increases, the high synchronization accuracy can still be achieved. Since the hierarchical structure covers the entire network based on a root node, the whole network can be synchronized to the same time reference. As a result, network-wide synchronization is possible. In addition, TPSN requires each node to exchange timing information with its parent in the hierarchical structure. Consequently, in TPSN, each node has to exchange synchronization information with only a single node and the protocol ensures that it is synchronized with all the remaining nodes in its neighborhood. Moreover, the synchronization cost is relatively low compared to NTP [96]. TPSN requires a hierarchical structure to exist for synchronization. The maintenance of this structure in the case of failed nodes increases the energy consumption. The hierarchical structure also prevents accurate synchronization of mobile nodes. Since the connectivity of different nodes changes as nodes move, the hierarchical structure needs to be formed accordingly. Moreover, the synchronization procedure of TPSN is based on adjusting the clocks according to the parent nodes in the hierarchy. This increases the cost of synchronization compared to other methods where the relative offsets of the neighbor nodes are stored and the time is translated without adjusting the physical clock. When multiple root nodes are used in large networks, each cluster can be synchronized to a different reference time. As a result, the protocol forms islands of times. To prevent this, each root node should be synchronized in advance. Of course, this increases the overall cost for synchronization if the root nodes are located far from each other. Furthermore, multi-hop synchronization is not supported since nodes synchronize only to their parent node.

2.4 Wakeup Mechanisms

WSN are energy-limited so typically the nodes cannot keep radios active during all the time, having to sleep and to wake up periodically [98]. Addressing this issue, there have been

proposed several MAC protocols which were categorized as *synchronous* or *asynchronous* MAC protocols. Although asynchronous protocols are simpler, they tend to consume more energy. But in WSN, where energy must be saved, a different approach may be used. One possibility is to use *synchronous* methods. Using these protocols, some techniques are adopted to increase the nodes lifetime: i) *duty cycling*; and ii) *scheduled rendezvous*.

2.4.1 Duty Cycling

Duty cycling is one mechanism widely used for energy-efficient MAC protocols in WSN. A MAC protocol that implements duty cycling, uses appropriate sleep/wake up mechanisms to conserve energy, and in [99] it is demonstrated that when sensor nodes remain in the sleep mode they consume less energy than when in the idle mode. When there is no need for communication, the radio is put to sleep and, although applying duty cycling energy is conserved, it has some disadvantages. Putting sensors into sleep mode makes it difficult to the all network to function, or at least certain part of it. As showed in [100], a few issues are needed to overcome such as deciding when to switch a device to low power mode or deciding *"for how long should a device remain in the low power mode?"*. To solve these issues, efficient and flexible duty-cycling techniques have been proposed. The S-MAC [101] and the T-MAC [102] protocols are examples of them. These protocols transmit a *SYNC* packet to notify neighbors about their schedule and to synchronize the clocks of all nodes in the network. The method only compensates for clock offset and does not consider clock drift [98]. Moreover, the knowledgement of traffic patterns can also help to take decisions about wake up. This method is known as adaptive duty cycling. S-MAC [101] is one of the major energy-efficient MAC protocols that efficiently exploits the idea of adaptive duty cycling. It uses a periodic sleep-wake up mechanism in order to lower power consumption. If a node has no packet to receive, it can waste a large amount of energy by just listen to the channel. Consequently, a node can save a significant amount of energy if it simply goes to sleep mode by switching off its radios [99]. T-MAC is an improvement over S-MAC duty cycling. In the T-MAC, listening period ends when no event has occurred for a time threshold *TA*. Though it improves on S-MAC, T-MAC has the disadvantage that it can face an early sleeping problem where a node can go to sleep even though its neighbor may still have messages for it. Synchronization is also an issue in duty cycling MAC protocols. In [103] is argued that synchronous MACs such as S-MAC have low energy consumption for sending packets but are complicated due to the need of synchronization. Conversely, asynchronous MACs, for example WiseMAC [104], is very simple, but it spends much energy in finding the neighbor's wake-up time. Moreover, synchronous methods can be characterized as one-way methods. Usually, the senders broadcast a reference message and receivers, upon the reception of the message, record the arrival time by their own clocks and exchange this information among each other to compensate clock offset between them. In [98] is proposed a synchronous method in which clocks in the all network are not modified. Instead, the nodes are synchronized with their own clocks. Since the periodic broadcast event in the network is the same, although they have different measurement results for this period by their own clock unit independently, they are able to interact with each other at

the same physical time. Without complicating the estimation process, and without modifying the clock of a node, this synchronization method becomes simpler and more energy-efficient than the traditional synchronization one-way method.

2.4.2 Scheduled Rendezvous

This type of MAC protocol requires a prescheduled *rendezvous* time at which neighboring nodes wake up simultaneously. In this method, a node wakes up periodically and sleeps until the next *rendezvous* time. A *scheduled rendezvous* scheme is shown in Fig. 2.26 [99].

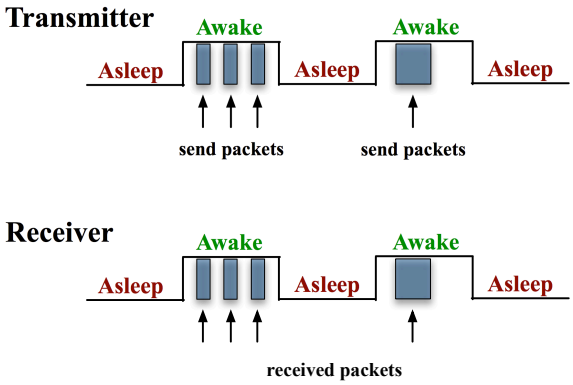

Figure 2.26: A scheduled rendezvous scheme

The advantage of this scheme is that when a node is awake it is guaranteed that all its neighbors are awake as well. Consequently, it is easier to send/receive packets. Broadcasting a message to all neighbors is also simpler in *scheduled rendezvous schemes*. RI-MAC [105] is a *receiver-initiated* asynchronous duty cycle MAC protocol for WSN. It uses a *receiver-initiated* data transmission in order to proficiently operate over a wide range of traffic loads. It attempts to minimize the time a sender and the receiver occupy the medium to find a rendezvous time for exchanging data, while still decoupling the sender and receiver's duty cycle schedules. A disadvantage of such MAC protocol is the requirement to maintain strict synchronization, because clock drifting may deeply affect the rendezvous time.

2.4.3 Discussion

The basic idea of low duty cycle protocols is to reduce the time a node is idle or spends overhearing an unnecessary activity by putting the node in the sleep state. The main goal is to put a node sleeping the most of the time and waking it up only when to transmit or receive packets. In the literature, low duty cycle protocols are classified as synchronous and as asynchronous low duty cycle MAC protocols. Synchronous low duty cycle MAC protocols typically comprise predetermined periodic wake-up schedules for data exchanges which consist of a sleep period and an active, repeated during the time in which the nodes are waked. On the other hand, asynchronous low duty cycle MAC protocols do not provide prior knowledge about the global or local timing information and schedules to the nodes in a network to assist with data

communications. Therefore, the nodes do not need to remember the schedules of its neighbors which significantly reduce the usage of memory and energy cost due to schedule sharing between the nodes. Besides the protocols above described, more exist and MR Ahmad et al. in [106] published a survey of low duty cycle MAC protocols in WSN.

With regard to *scheduled rendezvous* schemes, the basic idea is that each node should wakeup at the same time as its neighbors. Typically, nodes wake up according to a wakeup schedule, and remain active for a short time interval to communicate with their neighbors. Then, they go to sleep until the next rendezvous time. Different schemes differ in the sleep/wakeup pattern followed by nodes. According to [107], a drawback of the scheduled rendezvous schemes is that energy saving is obtained at the expense of an increased latency experienced by messages to travel through several hops. An additional drawback is that nodes must be synchronized. Besides the schemes described ind this section, also in the literature (e.g. [108]) several other clock synchronization protocols have been proposed to keep nodes synchronized. However, maintaining a tight synchronization among nodes requires a high overhead in terms of exchanged control messages. This, of course, results in energy consumption. The basic assumption behind scheduled rendezvous schemes is that the energy spent for keeping nodes synchronized is largely compensated by the energy saving achieved through power management.

2.5 Summary

Section 2.1 introduced a set of concepts related to WSN, including technologies and protocols that were used to achieve the goals of this thesis, namely the *LLN* concept, the IEEE 802.15.4 standard, the *6LoWPAN* adaptation layer, the RPL routing protocol, operating systems and simulation environments, with special focus on *ContikiOS* and *COOJA*.

Section 2.2 provided a detailed characterization on WSN energy efficiency.

Section 2.3 provided an overview on the techniques used to perform time synchronization for WSN.

Section 2.4 described wakeup mechanisms used to increase the nodes lifetime, namely the *duty cycling* and the *scheduled rendezvous* techniques.

Chapter 3

Application-Driven Wireless Sensor Network

WSN, being constituted by sensor nodes which are known to be energy constrained, depends on their nodes lifetime. Thus, in order to reduce nodes energy consumption, routing strategies capable of finding energy-efficient paths are demanded. We assume that routing protocols must find routes in which the nodes may be kept asleep the maximum amount of time they can. For that purpose, we use the concept of application duty cycle time, characterized by states wake and sleep, and their times. We also assume that WSN forms a mesh network, and that nodes may run multiple applications. Therefore, how to use mainly the nodes running the application associated to the data being transferred by the network, so that the nodes associated with other applications can continue sleeping? In this context, we define *Application-Driven WSN* (ADWSN) as a cross-layer solution aimed to help reducing the energy consumed by a network of sensor nodes executing a set of applications. This paradigm assumes that each application defines its own network and set of nodes so that the exchanged information can be confined to the nodes associated with the application. The nodes share information about the applications they run, and also their duty-cycles.

3.1 Constraining RPL-defined Routing Trees

RPL-BMARQ stands for ***RPL By Multi-Application ReQuest***. It tries to insure that data of an application is relayed mainly by the nodes running that application, i.e., *RPL-BMARQ* constrains RPL-defined routing trees. When sink nodes query the other nodes, routing paths should involve preferentially nodes running the same application. For that purpose, each query packet includes information about the associated application (*APPID*), which is known by the nodes running that application. Our routing scheme tries to insure that data of an application is relayed mainly by the nodes running that application. When the sink node queries the other nodes running the same

application, routing paths follow the DAG created. This DAG is created and maintained by a change in the RPL protocol scheme which will choose mainly the nodes running that application as parent; the nodes not associated to this application will not be selected as parent, in a first attempt. The nodes are put asleep when there is no activity related to their applications. When nodes receive a query packet they know exactly when they must wake up on the next period. In our solution the nodes alternate between the wake and sleep states. The amount of time of each phase is determined by the applications duty cycle. When a node is awake it performs activities including waiting for a sink query and forwarding packets to neighbors. When the wake up time expires, the node switches to the sleep state, waking up again by the time computed by the proposed synchronization mechanism described in Chapter 4.

3.1.1 Cross-layer Information

Cross-layer optimization in WSN has been addressed by multiple studies in different scenarios [109]. The main idea is to design communications layers such that they can share and react to information from other layers. In recent years cross-layer design was used to increase the efficiency of WSN communication systems, and Mendes et al. in [110] survey cross-layering techniques for WSN, which consider the *application*, *network*, *medium access control*, and *physical* layers interaction to design routing protocols because those layers attract special attention by their impact on the network lifetime. In fact, using cross-layering techniques, the layered structure of the communications stack can be altered to address many challenges that arise because of the interactions between the different layers.

RPL-BMARQ uses application layer information in order to create DAGs and to synchronize the nodes using a synchronous method mechanism. Each *ICMPv6 RPL DIO* message has information about: 1) the application the node runs and application duty-cycle, i.e. the time cycle of the application and the time the nodes are expected to be woken; 2) the number of neighbors; 3) the number of neighbors running this application. This information is used by a node to maintain its *neighbor table* (Fig. 3.1 represents an entry of this table), to maintain routing tables and to create and maintain DAGs. From a *neighbor table* a node knows its neighbors IPv6 addresses (IPADDR: global and LLADDR: link-local), what kind of node they are (TYPE: DAG root, sink, sensor, or other), what applications they run (APPID), and their correspondent duty-cycles (TCYCLE and TON). A node also knows the total number of neighbor it has (NBR1), and how many of them run its application (NBR2).

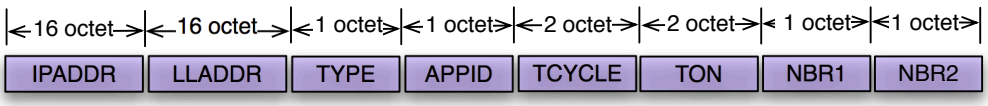

Figure 3.1: Example of tuple in the *RPL-BMARQ* neighbor information table

Fig. 3.2 shows the *DAG metric container object* used by *RPL-BMARQ* which needs to be included as an extension to any *RPL objective function metric container object*. A *DAG metric*

container object [12] is a *RPL control message option* used to compute the rank of a node, and to help the selection of the best parent. In *RPL-BMARQ*, this *metric object* includes information from the application layer with respect to: the identification of the application that the node runs (*APPID*), which may also be used to identify groups of applications with the same duty cycles so that multiple applications running with the same duty cycle can be included in the same DAG; the total application cycle time (*TCYCLE*); the total time the node is expected to be waked (*TON*); the number of neighbors that the node has (*NBR1*); and the number of neighbors that are running his application (*NBR2*). The *metric container object* is mandatory for the *RPL-BMARQ* solution because it carries all necessary information to create and maintain DAGs and *neighbor tables*.

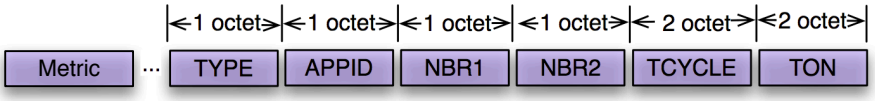

Figure 3.2: *RPL-BMARQ* metric container object

Each time an application layer query packet is sent by sinks the packet is "disseminated" into the network according to the *RPL-BMARQ* defined paths. This packet (see Fig. 3.3) is constituted by the *ORIGINATOR* field; *APPID* field; the *SEQNO* field; the *TTX* field; the *CMD* field; and by the *DATA* field. The *ORIGINATOR* field identifies the IPv6 Global Address of the packet originator; the *APPID* field identifies the application to which the packet corresponds; the *SEQNO* field is used to sequence a packet; the *TTX* field carries the timestamp (this time corresponds to the originator clock time); the *CMD* field specifies the type of the message (query, reply, or other); finally, the *DATA* field contains application data. This information is used not only to know if a packet is to be forwarded in the network layer, but also if it must be replied at the application layer. The *APPID*, *SEQNO* and *TTX* fields are also used by the synchronization mechanism to maintain the nodes synchronized, as described in Chapter. 4.

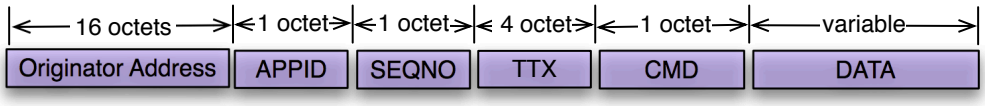

Figure 3.3: *RPL-BMARQ* application layer packet

This packet is sent through a particular UDP socket, as observed in the *RPL-BMARQ* communications stack (see Fig. 3.4), and included in the *data* field of a *transport layer* segment. Then, this segment is sent to the *network layer* using an IPv6 link-local address *FE80::* as destination. Since nodes can receive more then one query from neighbors, an incoming query buffer is used by the *RPL-BMARQ* routing mechanism which will help to decide if the received query is to be forwarded again to other neighbor nodes, to decide if the query is to be discarded in case of already have been received, and to decide if the query is to be replied back. The node consults its neighbor table to see if it has neighbors running the application from which the query was received. If there exists at least one neighbor, the query is forwarded again using the same link-local multicast

address (except if the query was sent by this neighbor); if not, the packet is discarded. Also, if this node runs the application from which the query was received, the datagram is passed up to the *transport layer* which will use the same UDP socket to send the query packet to the *application level*. At this level, upon the reception of the query message, the node processes it according to the information asked and sends back a reply by constructing a similar message packet (Fig. 3.3). This message is sent to the *transport layer* using another UDP socket, which will construct a new segment to be used by the *network layer* to send a datagram to the sink global IPv6 address. It has to be noted that queries are sent to a link-local IPv6 multicast address, whereas replies are sent to global IPv6 unicast addresses.

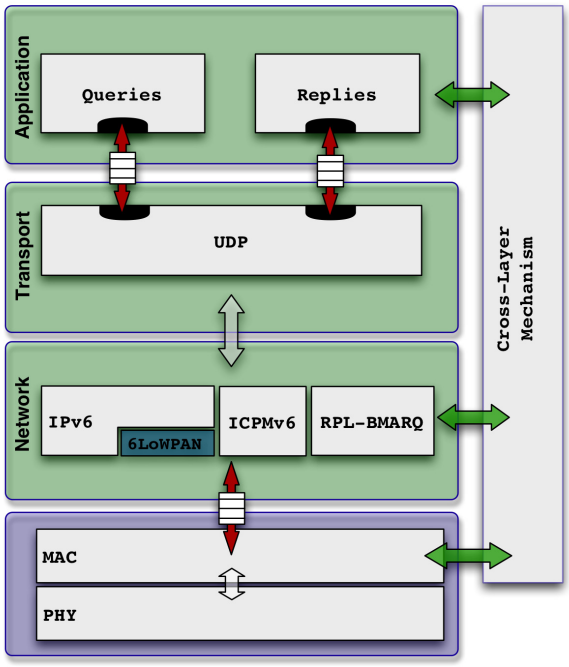

Figure 3.4: *RPL-BMARQ* communications stack

3.1.2 DAG Creation Mechanism

Let us assume that every node can participate in route discovery and packet forwarding. However, the nodes forwarding a given type of data will be primarily selected from the set of nodes running the same application to which the data is associated. For that purpose, each query packet includes information about the associated application (*APPID*), which is known by the nodes running that application. Our routing scheme tries to insure that data of an application is relayed mainly by the nodes running that application. When the sink node queries the other nodes running the same application, routing paths follow the *Directed Acyclic Graph* (DAG) created.

This DAG is created and maintained by our changes to the RPL protocol scheme which uses mainly the nodes running that application; the nodes not associated to this application will not participate in routing process, in a first attempt. Fig. 3.5 a) shows a network topology supporting two different applications. Fig. 3.5 b) shows the DAG created with standard RPL, and Fig. 3.5 c) shows the DAG created by our proposed solution.

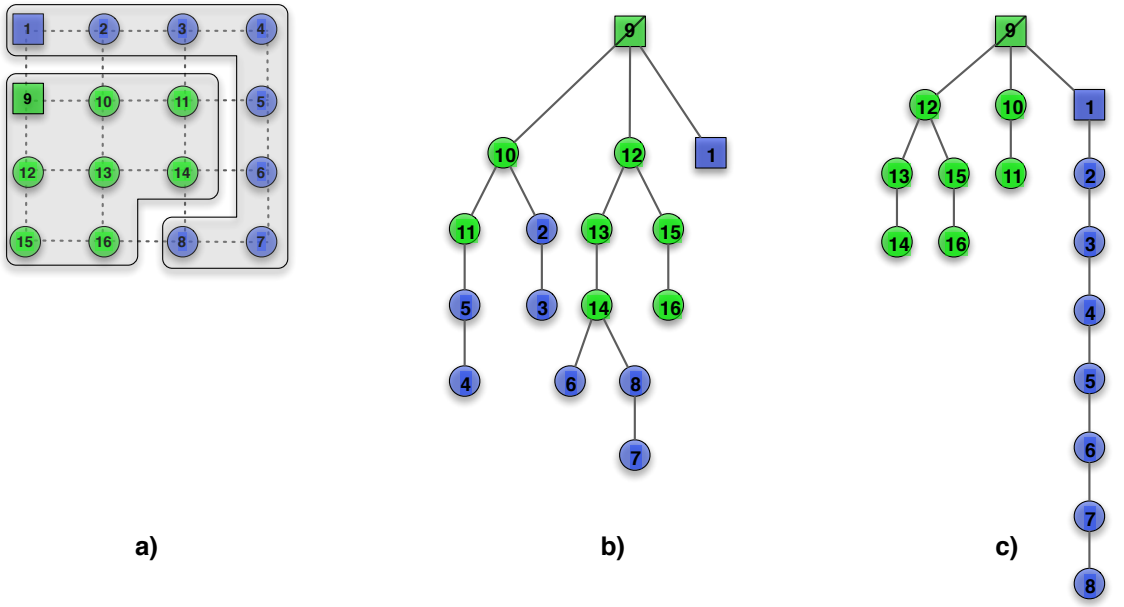

Figure 3.5: Application-Driven WSN concept. a) network topology; b) RPL DAG; c) RPL-BMARQ DAG

As it can be observed in the figure, the DAG creation scheme will use mainly the nodes running the same application; the nodes not associated to the application will not participate in DAG creation process, in a first attempt [14]. Algorithm 1 shows how to create DAGs in our solution. A root node starts to create the DAG by sending DIO messages. The nodes surrounding the root use the information carried in these DIO messages, compute their rank and join the DAG, in the same way as in regular RPL. The computed rank is jointly sent with the identification and duty-cycle of the application, in DIO messages. This information is used by other nodes to update their *neighbor tables*, compute their own rank, and advertise their presence by sending new DIO messages. All node's *neighbor tables* record the neighbors IP address, the application that they run and its duty-cycle. This information is used to help the node to join the DAG, by looking into its *neighbor table*. If the node runs the same application of its neighbor and if the latter has a lower rank, the former may choose it as parent, joining the DAG through it. If the node has neighbors which do not run its application, but have in turn at least one neighbor running this application, one of them can be selected as parent. If the neighbor has two neighbors, one root and the other sink, the root node will be always selected. Therefore, the node will not change parent depending on packets arrival. Otherwise, a node will always select a sink node neighbor as parent. If a node has only sensor nodes as neighbors it will select for parent the sensor running the same application which

has lower rank. Moreover, if the sensor has neighbors not running the same application but in turn they have other neighbors which run its application, the former will select as parent a neighbor with lower rank. In other situations, the mechanism switches to regular RPL. This mechanism is introduced inside a *RPL Objective function* to create the DAGs accordingly.

Algorithm 1: *Pseudocode of RPL-BMARQ DAG creation executed by a node*

```
if (neighbor == is_root) then
    add_parent(neighbor);
else
    if (neighbor == is_sink && neighbor.rank < node→neighbor.rank) then
        add_parent(neighbor);
    else
        if (neighbor.app_id == node.app_id && neighbor.rank < node→neighbor.rank)
        then
            add_parent(neighbor);
        else
            if (neighbor→neighbor.app_id == node.app_id && neighbor.rank <
            node→neighbor.rank) then
                add_parent(neighbor);
                reconfigure(neighbor(app_id, app_duty-cycle));
            else
                if (neighbor.rank < node→neighbor.rank) then
                    add_parent(neighbor);
                end
            end
        end
    end
end
```

3.1.3 Application-Driven Multicast Mechanism

The DAG creation mechanism described in above allows the creation and maintenance of the DAGs and routes, accordingly to our Application-Driven concept. Therefore, it is used mainly to give routing paths to unicast packets, being examples replies from sensor nodes to sinks queries. On the other hand, queries issued by sink nodes are forwarded (*"link-local multicast"*) in an application-driven way through the entire WSN. At this end, it is necessary that all multicast packets, which are sent on a link-local scope, are received mainly by the nodes which run the same application from the sinks which issue the query packet. Other nodes running different applications upon a possible reception of such queries should not forward them to their neighbors, unless they support the application.

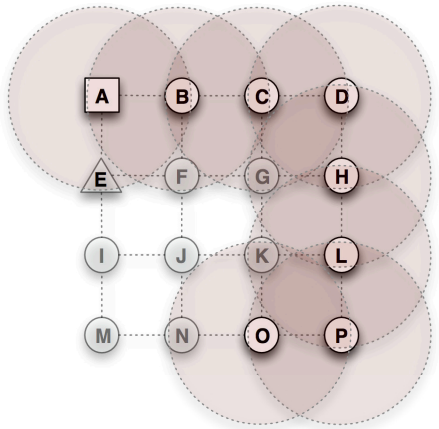

Figure 3.6: Application-Driven multicast explanation scenario

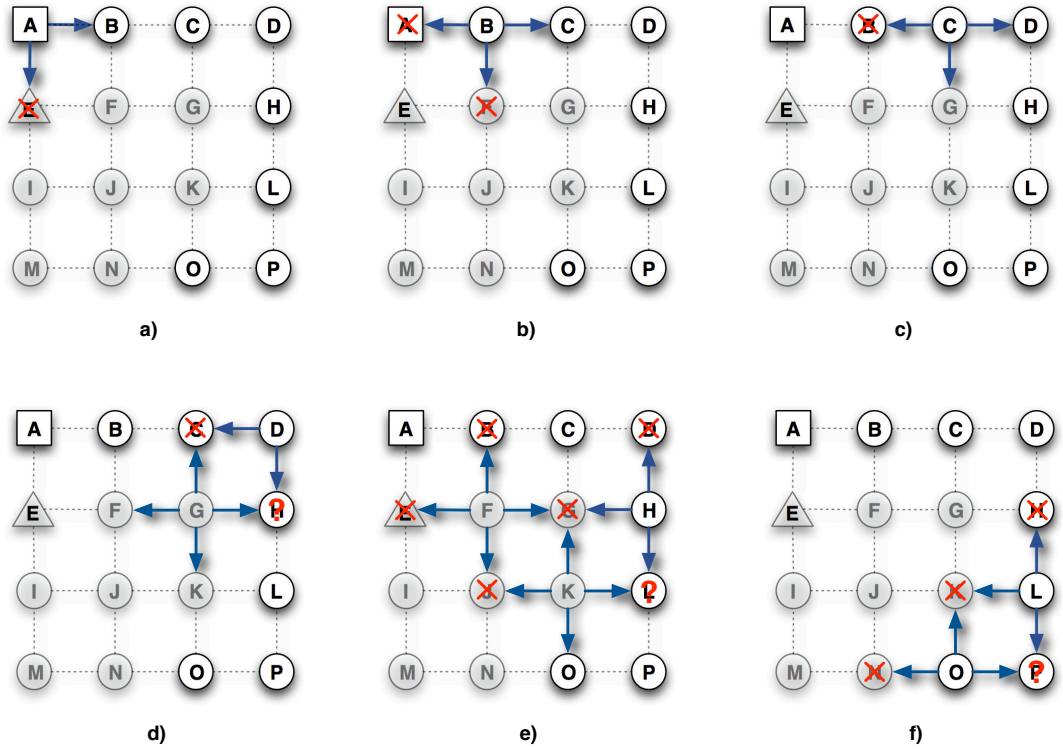

Figure 3.7: Application-Driven multicast explanation

Let us take as example Fig. 3.6. This figure represents a WSN with 16 nodes in a square-lattice 4x4 deployment, supporting two sensor nodes applications: nodes A, B, C, D, H, L, O and P run one type of the network supported applications (eg. App. A), and the other nodes run the other application (eg. App. B). In this example, when a query packet is issued, only the nodes B, C, D, H, L, O or P should forward the same packet in order to allow it to reach all the nodes running App. A (at least in a first attempt). The other nodes should not be involved in the forwarding process. At this end a *Application-Driven multicast mechanism* was designed considering above aspects. Fig. 3.7 demonstrates how in the above example this *multicast mechanism* should perform: In a), node A multicasts a query packet which should reach all the nodes in the network, except for the nodes not running this kind of application. Node A is the App. A sink node, and node E is at same time the DAG root and the sink node of App. B. When the query packet reaches node E, the mechanism should not forward the packet because node E does not run App. A and it does not have neighbors running that application, except for the originator of the packet (node A). Therefore node E should discard the packet. In b) when the query packet arrives at node B, the packet should be forwarded because this node runs the same application and it has at least one neighbor which also support the same application; on the other hand, when the packet arrives to node F, it will be discarded for the same reason as in a). The process continues, and in c) when the query packet arrives at node C, it will forward the same packet in order to let node D receive it; node B will discard the packet since it already has forwarded the packet; on the other hand, node G will not discard the query packet because it has at least one more neighbors which run App. A (node H), and it should get the query packet; in d) nodes D and G forward the query packet which will be discarded by node C since it has already be involved in the forwarding process, and node H should discard the duplicate query packet received (the last query packet sent by node D or node G); in e) the forwarding process continues and for the particular case of node F, since it has one neighbor node which runs App. A (node B), and it does not know that node B had already forwarded this query packet, node F will forward the query packet, which should be discarded by nodes E and J because they do not run this application and do not have neighbors who support it; nodes B and G will also discard the packet because they have been evolved earlier in the process; also, node K having at least two neighbor that support App. A will forward the packet, which should be discarded by nodes G and J; Node L should get the query packet from both node H and node K, and should discard the last packet received; finally, in f) node O receives the packet from node K and since it has one neighbor running his application (node P) it will forward the packet; node N should discard it, and node P will most likely receive two query packets, one from node L, and the other from node O, and since it has no more neighbor nodes which may be evolved in the forwarding process, the process ends here. At the end, the query packet issued by the sink node A has arrived to all the correspondent nodes.

In order to know which queries were already sent/forward and by whom, the mechanism needs a structure (Fig. 3.8) to record this kind of information which will be considered to make decisions by the *Application-Driven multicast mechanism* algorithm. The mechanism uses a table called ***Application-Driven multicast table*** (see Fig. 3.8) which records and updates the information

about the queries already sent/forward. Each tuple in the table contains information about the originator/forwarder of the packet (*ORIGINATOR* field), the application identification (*APPID* field) and the sequence identification (*SEQNO* field) of the packets. For each sensor nodes involved in the process, it should exist as many entries in the table as the query packets sent/forwarded.

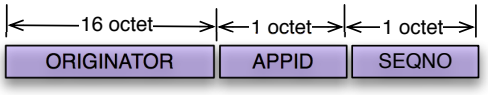

Figure 3.8: Application-Driven multicast table

Algorithm 2 shows the rationale of our *Application-Driven multicast mechanism*. Each node running the mechanism should go through the following steps: (1) verify if the packet is a multicast or a unicast packet (multicast packets are sent through link-local IPv6 addresses) - if the packet is a unicast packet, it should be processed by a *unicast process*; (2) being a query packet (multicast packet), and if the originator of the packet already exists in the multicast table, the entry is updated with the *SEQNO* information, else a new entry is inserted since that is the first query packet sent/forwarded by this originator node; (3) the mechanism should also verify if the originator of the packet is not the node itself, what means that the node received a packet originated by him and forwarded by a neighbor; (4) since it may be possible to the nodes to receive more than one query packet, possibly through several neighbors, the mechanism should verify if the packet was already been received by inspecting the sequence number and the application identification of the packet received, and by verifying into the multicast table if there exist more than one corresponding entry, even from different originators. If the sequence number of the entry packet is smaller than or equal to the query packet sent, then this means that the query packet was not yet been forwarded; (5) before forwarding the query packet the mechanism must verify if the packet meets the conditions required to be forwarded, i.e. the node has at least one neighbor running the query's application or, when not having one node in such condition, that it has at least one neighbor whose neighbors satisfy these conditions.

Algorithm 2: *Pseudocode of the Application-Driven multicast mechanism executed by a node*

```
foreach (packet_k received) do
    is_to_forward ← FALSE;
    if (packet_k.dest_address <> ll_address) then
        /* Process unicast packet!                                    */
        ucast_process(packet_k);
    else
        /* Process multicast packet!                                  */
        update_bmarq_mcast_table(packet_k);
        if (node.ll_address <> packet_k.orig_address)) then
            while (bmarq_mcast_table <> NULL) do
                mcast_entry ← lookup_bmarq_mcast_table(packet_k);
                if ((mcast_entry.seqno ≤ packet_k.seqno) && (mcast_entry.appid ==
                packet_k.appid) then
                    while (neighbor_entry ← lookup_neighbor_table() <> NULL) do
                        if ((packet_k.appid == neighbor_entry.app_id) ||
                        (neighbor_entry →neighbor_entry.app_id == packet_k.app_id))
                        then
                            is_to_forward ← TRUE;
                        end
                    end
                    if (is_to_forward) then
                        forward(packet_k);
                    end
                else
                    drop(packet_k);
                end
            end
        end
    end
end
```

3.2 RPL-BMARQ Evaluation

3.2.1 Validation Environment

Two different applications are used in our experiments. These applications and their topology are characterized by the following aspects: static, organized, pre-planned, no mobility, 16 nodes deployed in a square lattice topology, all sensor nodes battery-powered, except for the sink nodes. Additionally we consider multi-hop communications, the traffic pattern is point-to-multipoint when data is queried, and point-to-point when queries are replied by nodes. The first application (App. A) has a duty-cycle of one hour; every sensor node running this application wakes every hour remaining awake during one minute for receiving the query and send data back to the sink. The second application (App. B) has a duty-cycle of 15 minutes; the sensor nodes also wakes for one minute to sense and to send data to the sink and to communicate. As shown in Fig. 3.9 a), the period of App. A is 4 times the period of App. B. All sensor nodes are expected to be awake when data is queried and replied, and sleeping when there is no activity.

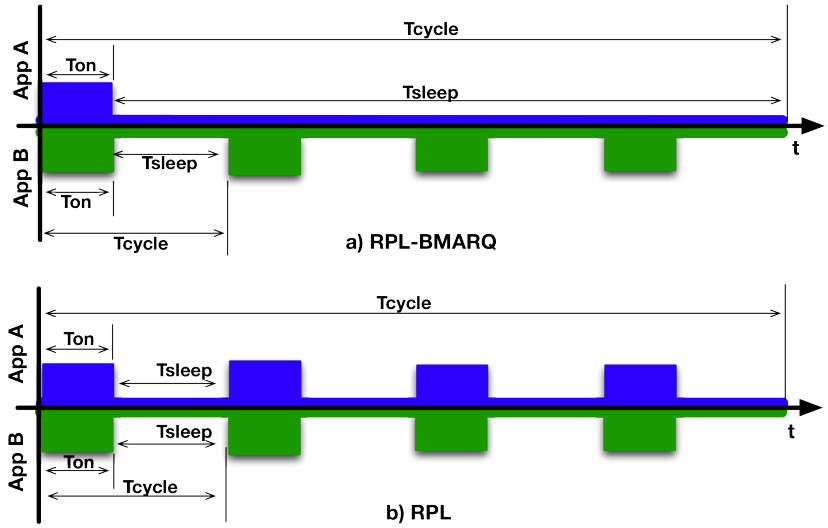

Figure 3.9: Applications activity cycle

In order to evaluate the *RPL-BMARQ* solution, a square lattice of 4x4 nodes was used. The nodes are distributed as shown in Fig. 3.10, where the four scenarios evaluated are also shown. All the nodes are within a distance of 25 meters for a transmission range of 30 meters, and support one of the two applications. Each application is running in eight nodes, and each node runs a single application. Sink nodes placements where chosen in order to allow long routing paths, since long paths consume more energy. In *Scenario 1* the nodes running App. A were selected in a way that a long path could be obtained. In *Scenario 2* both applications have the same node distribution; in these scenarios we aim to investigate the influence of the application duty-cycle in energy consumption. *Scenarios* 3 and *4* are used to investigate situations where at least one node from other application is required to relay data. In the scenarios simulated, sink nodes are always awake, and sink node running App. B (node 9) was chosen as DAG root because of

its application duty-cycle. Since our solution constrains the paths to the nodes associated to the application, a node needs to be woken up only when its applications run and not for generic routing and forwarding purposes. In contrast, RPL was used as shown in Fig. 3.9 b); in this case despite the applications having different periods, all the nodes would have to wake up every 15 minutes in order to process routing messages. In our solution, a query is *"multicasted"* only to the nodes associated to the application and not the entire WSN. When the application nodes reply, only the nodes running the application will send and forward *unicast* packets. So, using the *RPL-BMARQ solution*, the routing paths are chosen not only according to RPL objective functions, but also considering the nodes belonging to the application for which the paths are required, and the total number of neighbors a node has, at least in the first attempt. Fig. 3.10 c) shows a case of node deployment where some of the nodes of App. A (nodes 7 and 8) are unable to receive sink queries because they are isolated. Fig. 3.10 d) shows a *very particular* node deployment, where nodes 10, 13, and 16 running application B are unable to receive sink queries because they are isolated. In this case, node 8 would be selected to participate in the routing and forwarding process, being selected by node 16 as parent. This last scenario raises a routing issue and it is discussed separately in section 3.2.3.2.

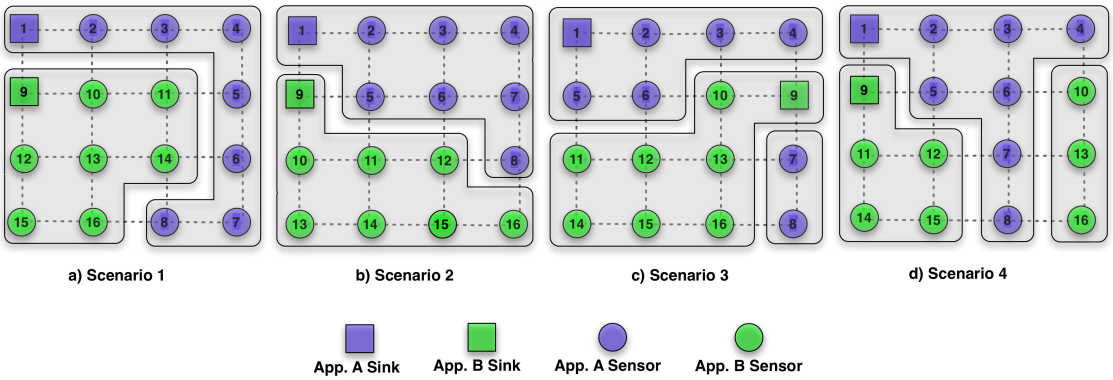

Figure 3.10: Nodes deployment in different square lattice mesh topologies

In order to evaluate the performance gains of our solution, first we estimate the magnitude of energy consumption improvements introduced by our solution. Then, we present and discuss the simulation results of the four scenarios and, finally, we present and discuss the results from two real testbed implementations.

3.2.2 Estimation of Energy Consumption

This study was focused on *Packet energy consumption*, considering the energy consumed when "broadcast", and "unicast" packets are sent and received by the nodes, and used to estimate *energy gains*.

3.2.2.1 DAGs Used

Fig. 3.11 shows the DAGs selected for theoretical evaluation. For *RPL-BMARQ* solution we have manually selected the DAGs so that the nodes may send and received packets as expected. For *RPL* the DAGs were selected in order to allow for shortest paths considering the hop count metric.

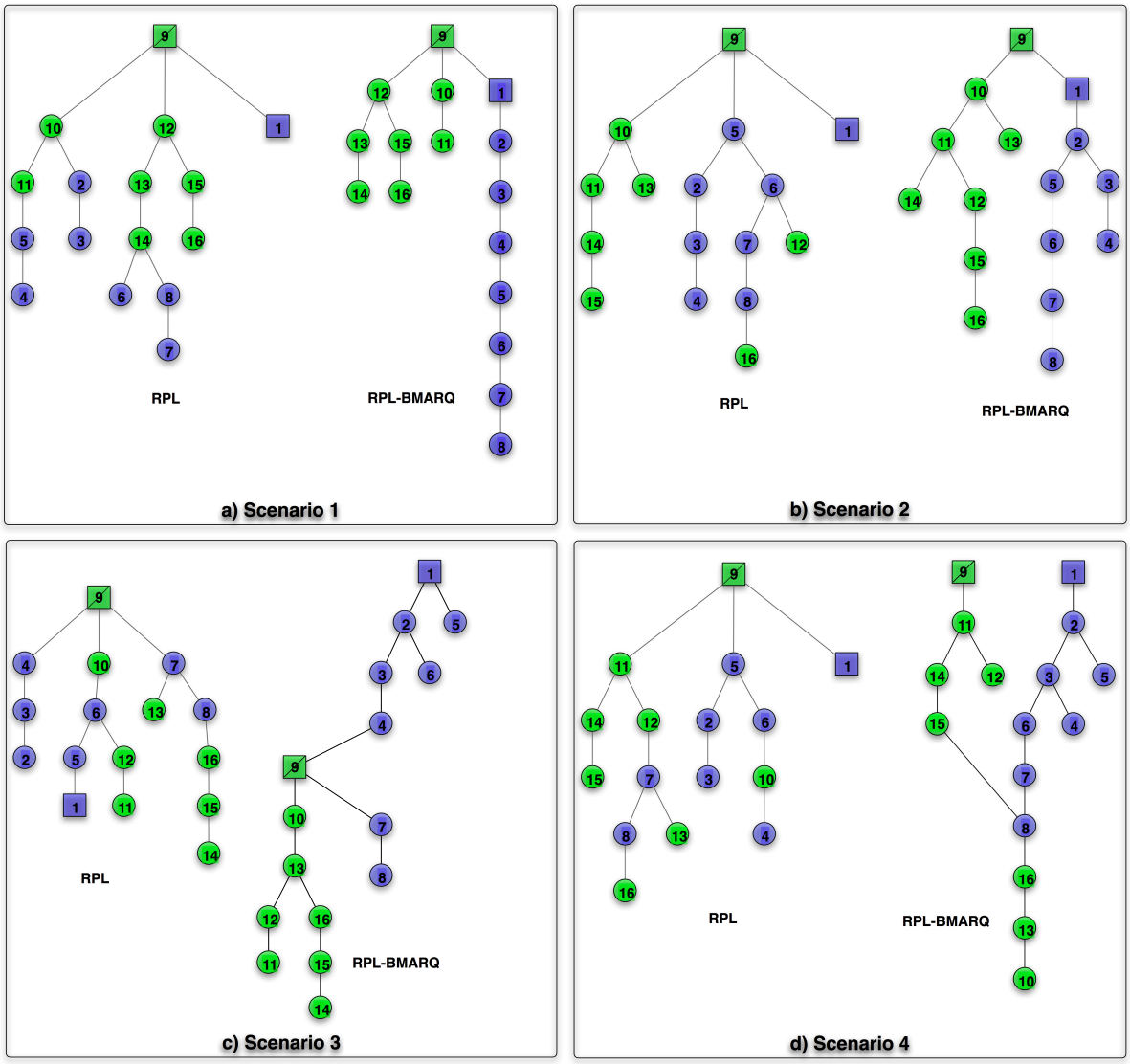

Figure 3.11: DAGs used in theoretical evaluation

3.2.2.2 Packet Energy Consumption

Considering that a node is implemented as a CrossBow TelosB [18] sensor hardware, and the energy is consumed by its CPU and RF transceiver, the total energy consumed can be described as

follows:

$$\begin{aligned} E &= E_{on} + E_{TX_{Bcast}} + E_{RX_{Bcast}} + E_{TX_{Ucast}} \\ &\quad + E_{RX_{Ucast}} + E_{Idle} + E_{Sleep} \end{aligned} \tag{3.1}$$

where E_{on} is the energy consumed during the time that node is waked, $E_{TX_{Bcast}}$ is the energy consumed when sending "broadcast" packets, $E_{RX_{Bcast}}$ is the energy consumed when receiving "broadcasted" packets, $E_{TX_{Ucast}}$ is the energy consumed when sending unicast packets, $E_{RX_{Ucast}}$ is the energy consumed when receiving unicast packets, E_{Idle} is the total energy consumed when the node is in the idle state (the state where a node has its radio on and waiting to send or to receive a data packet), and E_{Sleep} is the total energy consumed when the node is sleeping.

The energy consumed by a node in idle state is computed by $E_{Idle} = I_{Idle}$ (A) $\times V \times t_{Idle}$ (s), considering $I_{Idle} = 365\ \mu$A, $V = 3.6$ V , and t_{Idle} the time the node is idle, which depends on the communications scenario. The energy consumed by a sleeping node is computed as $E_{Sleep} = I_{Sleep}$ (A) $\times V \times t_{Sleep}$ (s), considering $I_{Sleep} = 5.1\mu$A, $V = 3.6$ V, and t_{Sleep} the total time the node is sleeping. The values are extracted from Table 2.2, obtained from [111].

We assume the simplest case of having no collisions and all the packets being correctly received. We also assume that a unicast packet is acknowledged at the MAC Layer, whilst a "broadcast" packet is not. The energy consumed per packet considering the information of Table 2.2, and the IEEE 802.15.4 specification [112], can be computed as follows.

- **Transmission of "broadcast" packet**: non-beacon enabled IEEE 802.15.4 networks use an unslotted CSMA-CA channel access mechanism [113, 114]. We assume that each time a device needs to transmit, it waits for a random number of unit backoff periods in the range $\{0, 2^{BE} - 1\}$ before performing the Clear Channel Assessment *CCA*. If the channel is found to be idle, the device transmits. If the channel is found to be busy, the device waits another random period before trying to access the channel again. Assuming the channel is found to be free, and also assuming that the backoff exponent *BE* is set to *macMinBE* which has the default value of 3, and the access time can be computed as

$$\begin{aligned} T_{CA} &= InitialBackoffPeriod + CCA \\ &= (2^3 - 1) \times aUnitBackoffPeriod + CCA \\ &= 7 \times 320\mu s + 128\mu s \\ &= 2.37ms \end{aligned} \tag{3.2}$$

  The *CCA* detection time is defined as 8 symbol periods. *aUnitBackoffPeriod* is defined as 20 symbol periods, where 1 symbol corresponds to 16 μs.

  As shown in Fig. 3.12, the energy consumed is computed as $E_{TX_{Bcast}} = E_i + E_{P_{TX}}$; E_i is the energy consumed during the *Channel Access* period (CA) which is $T_{CA} \times P_{Idle}$; P_{Idle} is the power consumed by the node in the idle mode which is 1.31 mW. $E_{P_{TX}}$ is the energy consumed during the time required to send the packet of size S (in octets). $E_{P_{TX}} = S \times T_{octet} \times P_{TX}$; T_{octet} is the time required to send one octet, which is 32 μs, and P_{TX} is the power consumed in the transmission of the same octet, which is 70.2 mW.

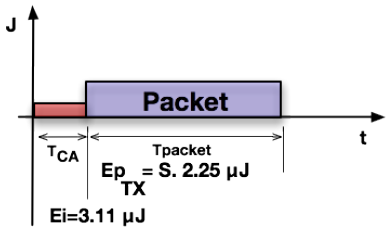

Figure 3.12: Energy consumed by a node when transmitting a "broadcast" packet of size S octets

- **Reception of "broadcast" packet**: when a node receives a "broadcast" packet of size S, the energy consumed is $E_{RX_{Bcast}} = E_{P_{RX}} = S \times T_{octet} \times P_{RX}$ where T_{octet} is the time required to receive one octet, which has the same value as the time required to send one octet. $P_{RX} =$ 78.5 mW is the power consumed by receiving the same octet, as shown in Fig. 3.13.

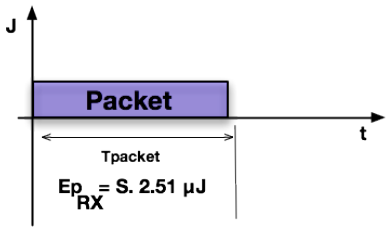

Figure 3.13: Energy consumed by a node when receiving a "broadcast" packet of size S octets

- **Transmission of unicast packet**: when a node sends a unicast packet, the amount of energy consumed is computed as the energy required to transmit the packet plus the energy consumed during the reception of the acknowledge frame. The transmission of an acknowledgment frame in a non-beacon enabled network commences *aTurnaroundTime* symbols after the reception of the data frame, where *aTurnaroundTime* is equal to 192μs. This gives the device enough time to switch between transmit and receive mode.

  As shown in Fig. 3.14, the total energy consumed is $E_{TX_{Ucast}} = E_i + E_{P_{TX}} + E_{T_{Ack}} + E_{M_{Ack}}$; E_i is the energy consumed during the *Channel Access* period; $E_{P_{TX}}$ is the energy consumed during the time required to transmit the packet of size S; $E_{T_{Ack}}$ is the energy consumed while waiting for the reception of the acknowledgment, which is 0.252 μJ, and corresponds to *aTurnaroundTime* times the power consumed in the idle mode which is 1.31 mW; $E_{M_{Ack}}$ is the energy consumed during the time required to receive the complete MAC acknowledgment frame which has a size of 11 bytes, and corresponds to 27.6 μJ.

- **Reception of unicast packet**: the energy consumed to receive a unicast packet of size S is $E_{RX\ _{Ucast}} = E_{P_{RX}} + E_{T_{Ack}} + E_{M_{Ack}}$. $E_{P_{RX}}$ is the energy consumed during the time needed to receive the packet of size S; $E_{T_{Ack}}$ is the energy consumed during the T_{Ack} time to wait before sending the acknowledge packet (this value is the same as the waiting time before receiving the acknowledge packet); $E_{M_{Ack}}$ is the energy consumed during the time of the transmission

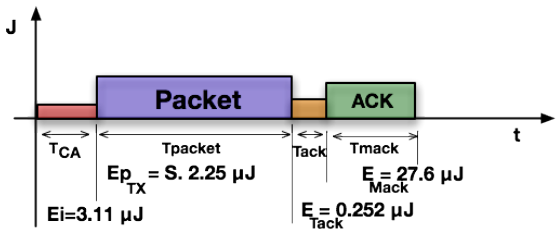

Figure 3.14: Energy consumed by a node when transmitting a unicast packet of size S octets

of the *acknowledge MAC frame* - T_{Mack} times the power consumed in the transmission mode which is 70.2 mW, corresponding to 24.7 μJ, (see Fig. 3.15).

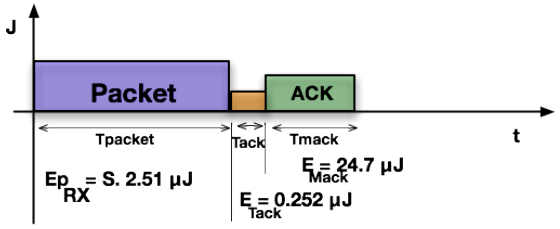

Figure 3.15: Energy consumed by a node when receiving a unicast packet of size S octets

- **Energy gain estimation**: we define the energy gain of our routing solution as:

$$\frac{E_{RPL} - E_{BMARQ}}{E_{RPL}} \times 100 \tag{3.3}$$

where E_{RPL} is the total energy consumed by the nodes for the RPL routing solution, and E_{BMARQ} is the total energy consumed by the nodes for our routing solution. The values are given in %.

3.2.2.3 Results and Discussion

We characterized the energy consumed by the nodes running simultaneously both applications, we take into account the number of *"broadcast"* (more exactly, *link-local multicast*) and unicast packets, and the time the nodes are wake, sleeping, or in idle mode during one hour. We also compare our solution against the generic RPL routing solution. The analysis was performed based on packets of 127 octets (application data size of 81 octets, plus the IEEE 802.15.4 MAC layer header and PHY layer header size of 46 octets). The packet size chosen reflects worst cases in the analysis performed. MAC layer collisions were not considered. For simplicity we assumed that all packets were sent and received with no errors and no retransmissions. The calculus was made using a C program that implements both solutions.

- **Energy**: Fig. 3.16 shows the total energy consumed by each node. As can be seen, our solution always consumes less energy. The total of energy consumed is computed as the sum of the energies consumed by individual nodes. The energy consumed by each node is given

by Eq. 3.1, and Fig. 3.17 shows the total of energy gains using de RPL-BMARQ solution in each scenario. The gain is computed using Eq. 3.3. Fig. 3.18 shows the energy consumption by each solution in the scenarios studied. As it can be seen, nodes using the *Standard RPL* solution always consume more energy. In the selected scenarios the mean energy consumed for the RPL-BMARQ solution considering the 4 scenarios, is *5.95 J*, and for the RPL solution is *8.76 J*. The mean gain, considering the 4 scenarios, is *32.7%*. Concerning to the time the nodes are sleeping, results show that for the RPL-BMARQ solution, nodes sleep more time than for the RPL solution. For the selected scenarios, and for the RPL-BMARQ solution, the sum of time the nodes are sleeping is 57,000 s, while for the RPL solution is 56,640 s. In the idle mode energy is also consumed.

- **Application packets transmitted and received**: Fig. 3.19 shows the distribution of the "broadcast" packets generated by each node in the four scenarios. As shown, the number of "broadcast" packets sent using our solution is lower than the number of packets sent using the RPL solution. In scenarios 3 and 4, the *RPL-BMARQ solution* needs node 9 and node 8, respectively, to forward packets. Those nodes forward 5 "broadcast" packets in scenarios 3 and 4. The total number of "broadcast" packets sent in each scenario corresponds to the packets sent when sink nodes issue queries. In our solution, only the nodes belonging to the application forward the packets, thus network flooding is bounded to the nodes of the application. In the case of the RPL solution, when a sink issues a query the packets are "broadcasted" to the entire network. Since there are two applications running, the network is "broadcasted" twice. For example in scenario 1 our solution sends 40 packets, while the RPL solution sends 80 packets.

  Fig. 3.20 shows the distribution of "broadcast" packets received by each node. As shown, the number of "broadcast" packets received with our solution is lower than the number of packets received using the RPL solution. For the RPL solution, in the scenarios considered, 4 nodes are used more often (nodes 5, 6, 7, and 12). They are placed in the middle of the topology and receive more packets than the others, since they have more neighbors. With the RPL-BMARQ solution, the distribution of the nodes inside the topology has influence. Looking at scenario 1, node 10 receives more packets than the others nodes (16 packets). This node runs application B, thus it is used 4 times per hour. It also receives more packets since it has more neighbors. In the case of scenario 4, there are 2 nodes (11, 15) which receive respectively 12 and 8 packets, because node 8 relays packets from nodes running application B. When analyzing Fig. 3.19 and 3.20, we also conclude that with the RPL-BMARQ solution application B generates more "broadcast" packets than application A, since queries are four fold. With RPL, all nodes are waked in order to receive and to send "broadcast" packets, so the node distribution does not influence the number of "broadcast" packets. Moreover with RPL, queries are issued 5 times per hour (one from application A, and 4 from application B). Fig. 3.19 and 3.20 also show "broadcast" *"hotspots"*.

  Fig. 3.21 shows the distribution of unicast packets sent by each node in the four scenarios

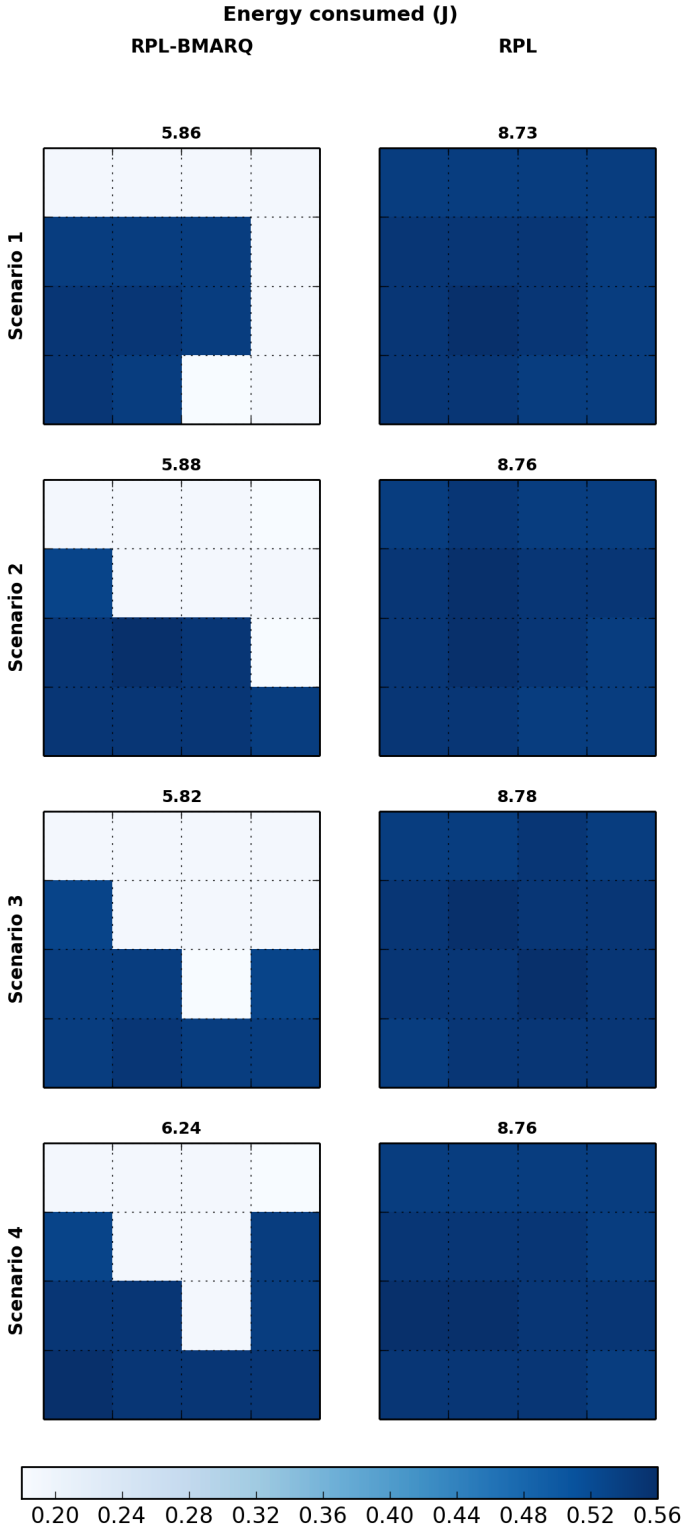

Figure 3.16: Total of energy consumed in each scenario for the RPL-BMARQ and the RPL solutions

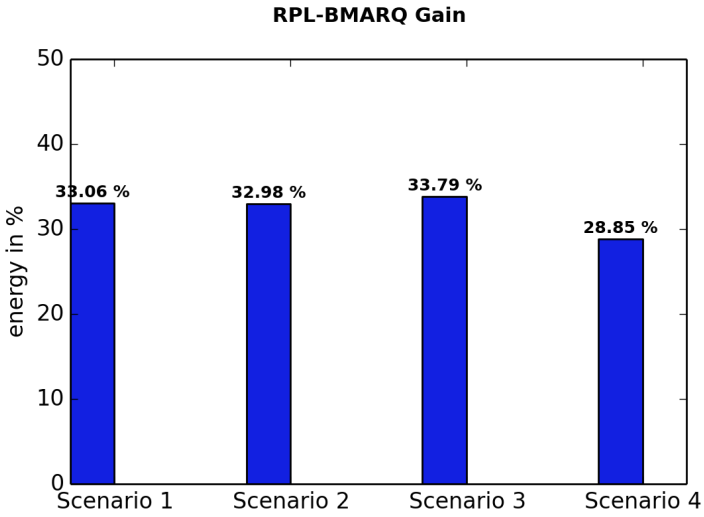

Figure 3.17: RPL-BMARQ energy gains in each scenario (in %)

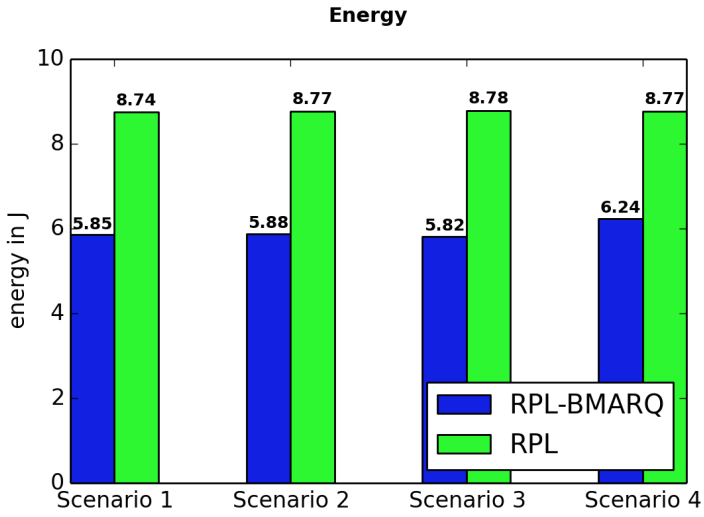

Figure 3.18: Energy consumption in each scenario (in J)

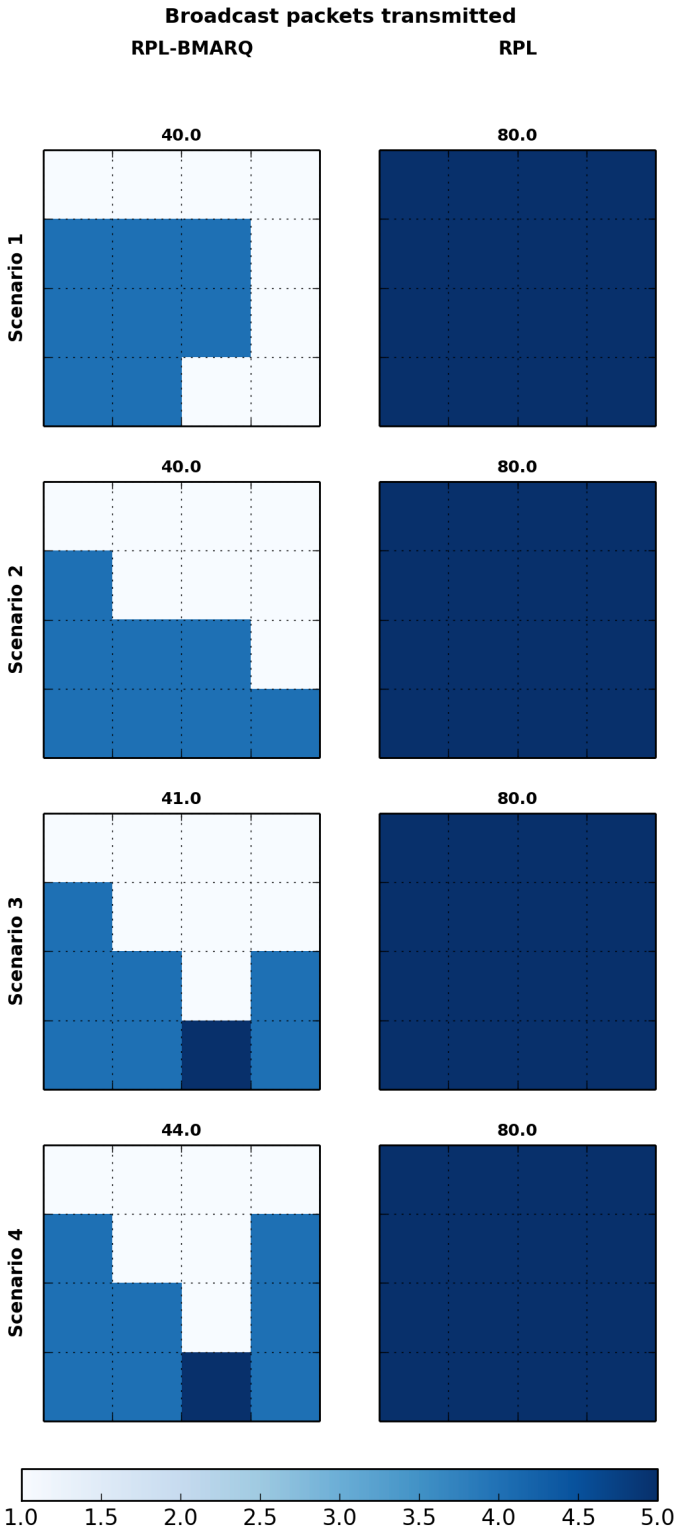

Figure 3.19: Total number of "broadcast" packets generated in each scenario for the RPL-BMARQ and the RPL solutions

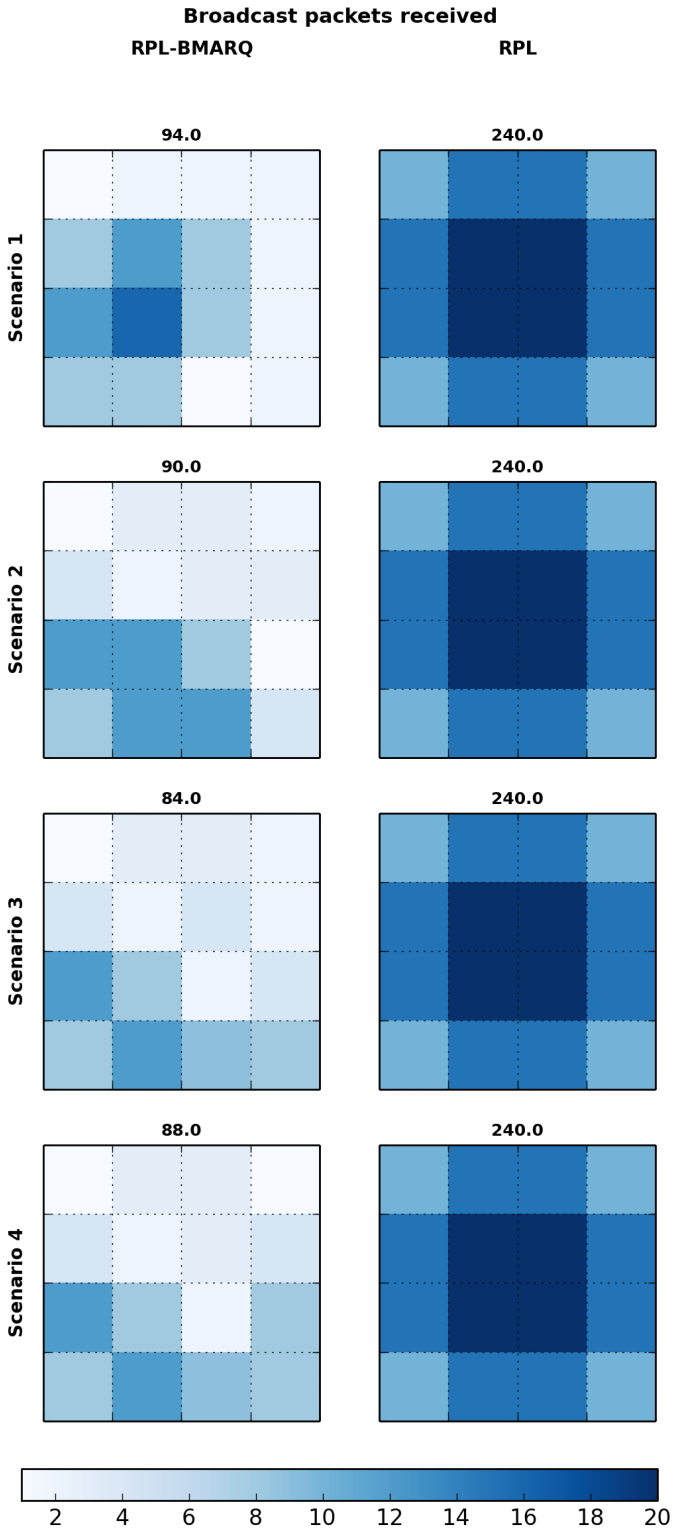

Figure 3.20: Total number of "broadcast" packets received in each scenario for the RPL-BMARQ and the RPL solutions

Table 3.1: Theoretical results

	Scenario 1				Scenario 2				Scenario 3				Scenario 4			
	BMARQ		RPL		BMARQ		RPL		BMARQ		RPL		BMARQ		RPL	
	Tx	Rx	Tx	Rx	Tx	Rx	Tx	Rx	Tx	Rx	Tx	Rx	Tx	Rx	Tx	Rx
BCast (packets)	40	94	80	240	40	90	80	240	41	84	80	240	44	88	80	240
UCast (packets)	84	191	90	88	100	249	106	347	51	119	120	383	124	262	106	344
Time waked (s)	600		960		600		960		600		960		645		960	
Time sleeping (s)	57,000		56,640		57,000		56,640		57,000		56,640		56,955		56,640	
Time idle (s)	597.43		955.88		596.93		955.35		598.21		955.01		641.66		955.37	
Energy consumed (J)	5.86		8.73		5.88		8.76		5.82		8.78		6.24		8.76	
Energy Gain (%)	33.0				33.0				33.8				28.9			

considered. We can verify that the total number of unicast packets sent in each scenario is lower for *RPL-BMARQ* than for *RPL*, except for scenario 4, where *RPL-BMARQ* needs to send more unicast packets. In this scenario nodes need to forward more packets in order to reply to sink queries. Since RPL-BMARQ uses mainly the nodes belonging to the applications, node 11 is more often used. In the scenarios considered, and analyzing the DAGs used (see Fig. 3.11) the paths selected by the RPL-BMARQ solution are longer than those selected by the RPL solution, which makes no distinction between nodes. In the case of scenario 4, node 8 is also used to forward packets from nodes running other application. In this situation, node 8 transmits 13 packets. Since for the RPL solution all nodes must be awake, in scenario 4, node 11 is the node with more activity, and it transmits 26 packets per query. The number of the unicast packets received in our solution is smaller than in RPL, as shown in Fig. 3.22). As example, in scenario 1 the total number of unicast packets received for the RPL-BMARQ solution is 191, while for the RPL solution is 290. The results are summarized in Table 3.1. They show that, for the scenarios studied, our solution provides significant energy gains.

- **Analysis for large networks**: the above theoretical results were obtained for small topologies. But one could ask how the proposed solution could lead to non-optimal path when large networks are involved. Let us take as example a large network, characterized by a square lattice topology of 100 nodes and consider two limit node distributions: i) 50 nodes running application A and 50 nodes running application B (Fig. 3.23 a), and ii) 10 nodes running application A and 90 nodes running application B (Fig. 3.23 b). These topologies represent large WSNs with different node distributions by applications (ratios 1/1 and 1/9) and, simultaneously, the usage of several nodes of running one application being used to relay packets generated by nodes running the other application. To perform this study we used the same method employed in Sec. 3.2.2.3.

  Results are summarized in Table 3.2. As it can be observed, in both topologies the energy consumed by *RPL-BMARQ* is always lower than the energy consumed by *RPL*. In topology a) and topology b) we can observe that our solution presents gains respectively of 30% and 90%. This difference comes from the sensors nodes distributions: in a) the application distribution ratio is the same (same number of sensor nodes for each application); in b) 90%

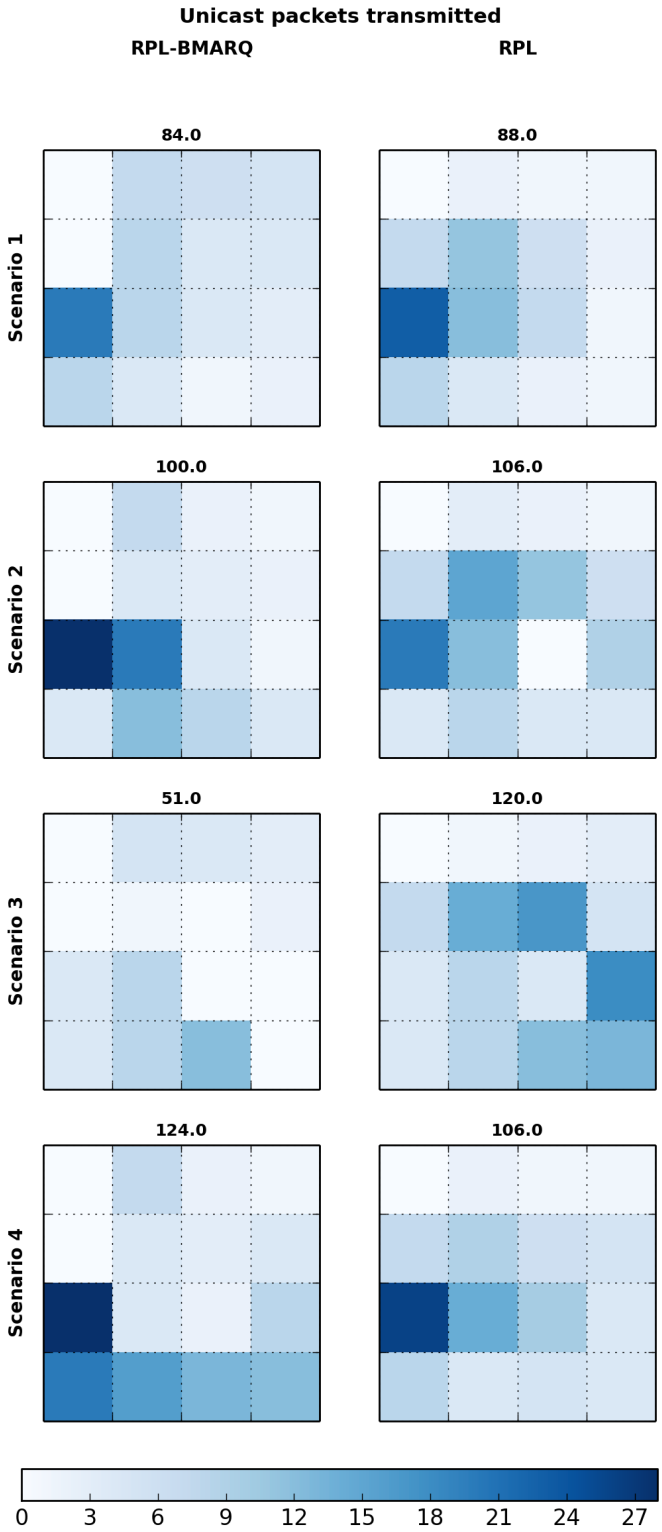

Figure 3.21: Total number of unicast packets transmitted in each scenario for the RPL-BMARQ and the RPL solutions

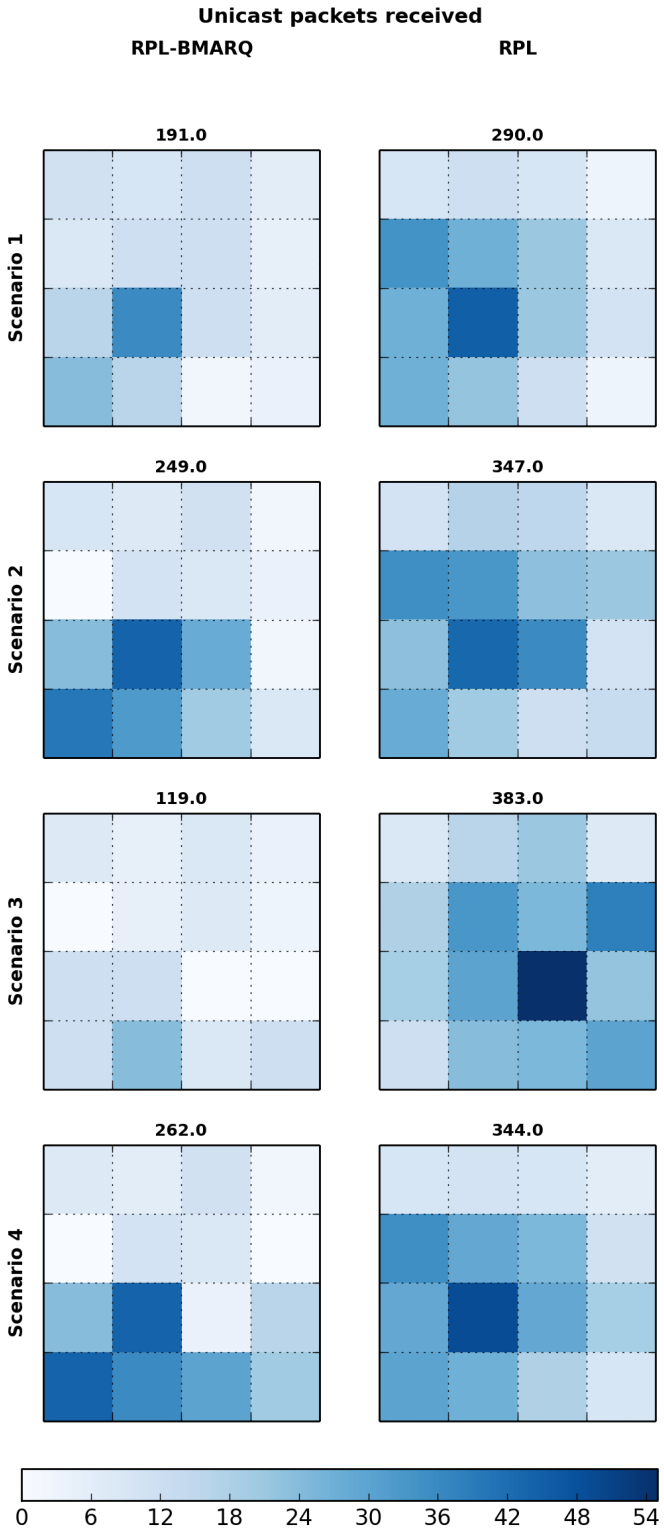

Figure 3.22: Total number of unicast packets received in each scenario for the RPL-BMARQ and the RPL solutions

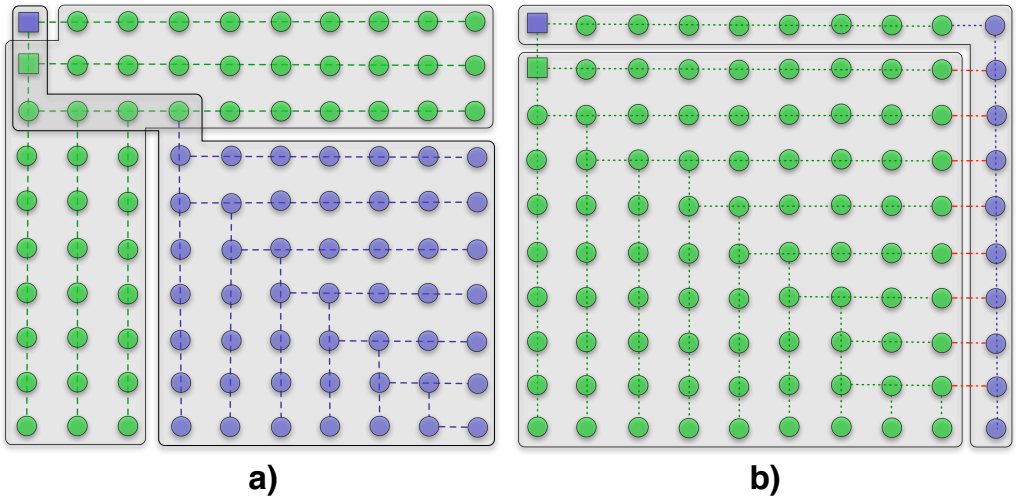

Figure 3.23: Impact on large networks: a) 50 nodes running application A and 50 nodes running application B; b) 10 nodes running application A and 90 nodes running application B

of the sensor nodes run one application which is four times grater then the duty cycle of the application running in the other 10% of the nodes.

Analyzing the total number of layer 3 packets sent and received, we observed that the *RPL-BMARQ* solution presents lower values than *RPL*. In a) the total number of *broadcast* packets received by *RPL-BMARQ* and *RPL* are respectively 848 and 930; the total number of *unicast* packets received by *RPL-BMARQ* and *RPL* are respectively $5,339$ and $5,998$. With respect to topology b) the total number of *broadcast* packets sent by *RPL-BMARQ* and *RPL* are respectively 379 and 500; the total number of *broadcast* packets received by *RPL-BMARQ* and *RPL* is are respectively $1,324$ and $1,800$; The total number of *unicast* packets sent by both solutions is the same $(5,607)$ and the total number of *unicast* packets received by *RPL-BMARQ* is lower $(20,953)$ than *RPL* $(21,109)$.

From these results we can conclude that for large WSNs the number of hops does not affect our solution as it still provides significant energy gains.

Table 3.2: Results obtained for large networks

	Topology a)				Topology b)			
	BMARQ		RPL		BMARQ		RPL	
	Tx	Rx	Tx	Rx	Tx	Rx	Tx	Rx
BCast (packets)	255	848	255	930	379	1,324	500	1,800
UCast (packets)	1,707	5,339	1,707	5,998	5,607	20,953	5,607	21,109
Time waked (s)	14,700		23,520		22,200		324,000	
Time sleeping (s)	338,100		336,000		203,400		336,000	
Energy consumed (J)	28.24		40.05		42.42		441.66	
Energy Gain (%)	29.5				90.4			

3.2.3 Simulations

In order to study the behavior of the RPL and the *RPL-BMARQ* solutions, both have been implemented in *ContikiOS* [16], which is an operating system used for wireless sensor networks. *ContikiOS 2.6* was chosen because it includes an IPv6 stack with *6LoWPAN* support, as well as *ContikiRPL*, which is a basic RPL implementation. The communication between the nodes is achieved using the *CSMA/CA* access scheme with *NullRDC*, which means that the *Mac Layer*, by it-self, does not put the nodes in the *on* and *off* states. The simulations were performed using *Cooja* [17], a simulator for *ContikiOS*, which allows developers to test code and systems before running it on a target hardware. The hardware platform selected was *TelosB* [18] which uses IEEE 802.15.4 radios. The 2.4 GHz radio model chosen was the *Unit Disk Graph Medium* (UDGM). UDGM models the transmission range between the nodes as an ideal disk; the nodes outside it do not receive packets, while the nodes within the transmission distance receive all the messages. *UDGM* confirms the functionality and behavior of our solution. Packets have a IPv6 Payload of 82 bytes, except for signaling.

Table 3.3 shows the code-size in bytes for sink and sensor nodes for *RPL* and for *RPL-BMARQ* solutions. As it can be verified, code-size overhead resulting from *RPL-BMARQ* implementation is about 7.1% for sinks, and 6.4% for sensor nodes. In contrast, code in RAM size is reduced by about 9.6% for sinks and 10% for sensor nodes.

Table 3.3: Code-size for RPL and RPL-BMARQ solutions. Shown is ROM (.text) and RAM (.bst + .data) in bytes

	RPL			**RPL-BMARQ**		
	ROM	**RAM**		**ROM**	**RAM**	
Sink	42280	186	7400	45492	186	6688
Sensor	42260	186	7400	45170	186	6662

3.2.3.1 Results and Discussion

We have simulated *RPL-BMARQ* solution in two situations. The first corresponds to a situation where all the nodes join the network at same time, so that the designed nodes synchronization mechanism is not used, being represented in figures as *BMARQ (no sync)*. In the second situation, the nodes will join the network at different time, and it is represented in the figures as *BMARQ (sync)*. The later implies the use of our second contribution, the synchronization mechanism described in Chapter 4, in order to keep the nodes synchronized with respect to the applications they run. The nodes join the network at different times which was randomly generated, and the expected random time value for a node to join a network is defined as

$$\begin{aligned} E[t_c] &= \tfrac{1}{2} \cdot Max(T_{Cycle_A}, T_{Cycle_B}), \\ t_c &\in [0, Max(T_{Cycle_A}, T_{Cycle_B})]. \end{aligned} \tag{3.4}$$

For the evaluated scenarios, T_{Cycle_A} equals 1 hour and T_{Cycle_B} 15 min. So, the average delay for a node to join the network is 1800 sec. Table 3.4 shows the time required to boot the network in these conditions.

Table 3.4: Nodes association time randomly generated

Scenario	Node type	Application	Node	Boot (in sec.)
1	Sensor	A	2, 3, 4	561
			5, 6, 7, 8	1027
		B	10, 12, 15	940
			11, 13, 14, 16	1102
2		A	2, 3, 5, 6	561
			4, 7, 8	1027
		B	10, 11, 13, 14	940
			12, 15, 16	1102
3		A	2, 3, 4, 5	561
			6, 7, 8	1027
		B	10, 11, 12, 13	940
			14, 15, 16	1102
4		A	2, 3, 5	561
			4, 6, 7, 8	1027
		B	11, 12, 14, 15	940
			10, 13, 16	1102
All	Sink	A	1	399
		B	9	317

- **DAGs created**: Fig. 3.24 shows the DAGs generated during simulations. For each scenario the simulations ran constructed the same DAGs. In this figure it can be verified that *RPL-BMARQ* solution constructs the DAGs as expected. A particular attention must be given to scenario 4 where node 16 choses node 8 as parent which does not run its application. In this situation, node 8 should not send packets from nodes 10, 13 and 16 to its parent (node 7), but use other link, sending them directly to node 15 which will forward them so that the node 9 (sink) can receive the packets. This aspect raised some issues and therefore it is discussed separately.

- **Energy consumption**: operating systems for wireless sensor networks such as ContikiOS [16] reduce energy consumption by powering off the microcontroller and hardware components when they are not used. The on-line energy estimation mechanism [89] defines the total energy consumption of a sensor node, E (J), as

$$\begin{aligned} E &= (I_m \times t_m + I_l \times t_l + I_t \times t_t + \\ &\quad I_r \times t_r + \textstyle\sum_{i=1}^{n} I_{c_i} \times t_{c_i}) \times V \end{aligned} \tag{3.5}$$

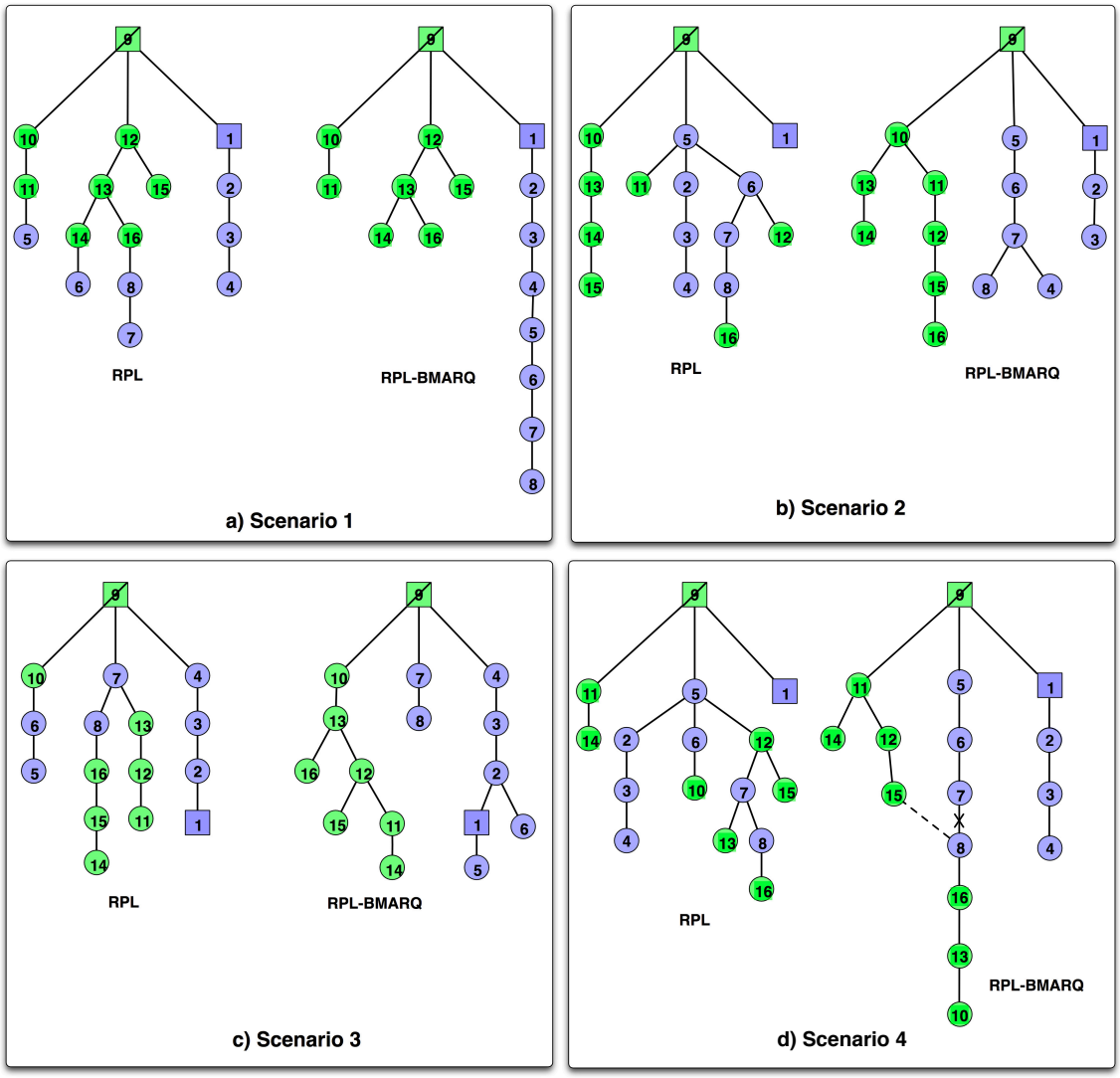

Figure 3.24: DAGs generated during simulations

being I_m the current consumed by the microprocessor in the time t_m during in which the microprocessor is running, I_l and t_l the current and time when the microprocessor is in low power mode, I_t and t_t the current and the time when the device communication is in transmit mode, I_r and t_r the current and the time when the device communication is in receive mode, I_{c_i} and t_{c_i} the energy consumed by other components (eg. LEDs, ADCs, DACs), and V the sensor supply voltage.

We consider energy consumption related only to communication aspects, e.g. packet transmission, reception, *radio "idle"* the state where a node has its *radio on* and is waiting to send or to receive a data "packet", and *radio interferences*. We aim to investigate how much energy is consumed by the nodes in these states, as shown by Eq. 3.6.

$$E = E_{TX} + E_{RX} + E_{Idle} + E_{Int} \tag{3.6}$$

E_{TX} is the total energy consumed when sending packets, E_{RX} is the total energy consumed when receiving packets, E_{Idle} is the total energy consumed by a node when it has the radio in the *idle state*, and E_{Int} is the total energy consumed by a node when it suffers radio interferences from it's neighbors. Generically, we compute energy E_{state} as $I_{state} \times V \times t_{state}$, considering I_{state} the current consumed by the node in the $state$, V the voltage supplied to the node, and t_{state} the total time the node is in that $state$. The I_{state} and V values depend on existing platforms (e.g. TelosB [18]), and a generic energy model defining the total energy consumed by the nodes is detailed in [13]. From the results obtained, we extracted data related to communications time which is shown in Fig. 3.25. This figure shows the mean, and 25, 50 and 75% percentiles of relevant radio state times. As one can observe, the permanence time in each state is lower for *RPL-BMARQ* (both *no sync* and *sync* implementations) than for *RPL*. Using *RPL-BMARQ*, the time where radios are in the idle state is less then that using *RPL*, which reflects the major contribution of *RPL-BMARQ* design. Applying the information of Table 2.2, which was extracted from [18], to Eq. 3.6, we can compute the total energy consumed by the nodes (see Fig. 3.26). Results show that the total energy consumed by the nodes considering those states, and using the *RPL-BMARQ* solution using both implementations is very low, as it turns the node's radio *off* during more time than *RPL*. Using Eq. 3.3 we compute the energy gain for *RPL-BMARQ* and results show that, using the *RPL-BMARQ* solution not implementing the synchronization mechanism, the nodes spend about 92% less energy than the same nodes running *RPL*. In the case of the second implementation of *RPL-BMARQ* in which the synchronization mechanism is used, the nodes spend about 85% less of energy than *RPL*. The difference in gain between the two implementations of *RPL-BMARQ* is due to the excess time the nodes in the synchronized implementation need to be awaked, thus spending more energy. Even though, this enables us to conclude that *RPL-BMARQ* extends the network lifetime.

- **Query Success Ratio**: *Query Success Ratio* (QSR) is defined as the ratio between the

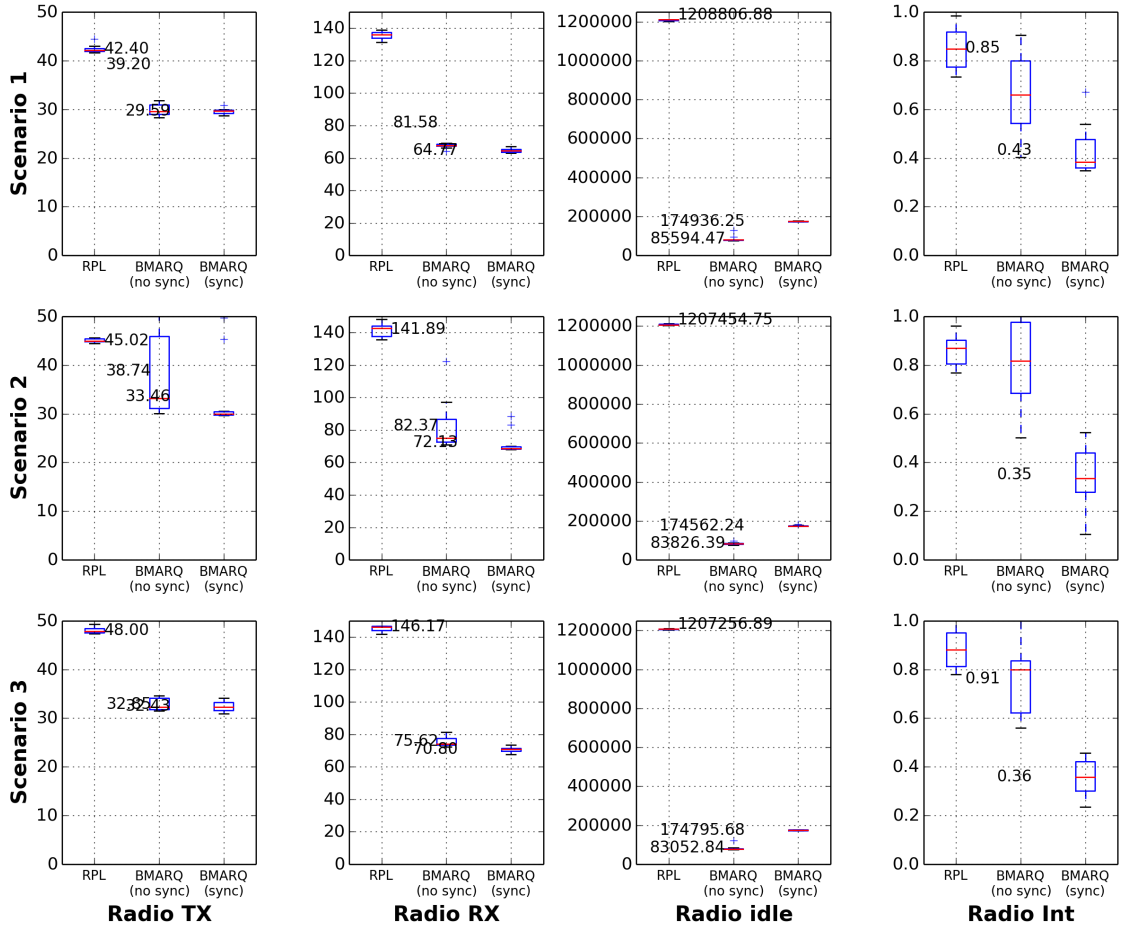

Figure 3.25: Radio Activity Times (in seconds)

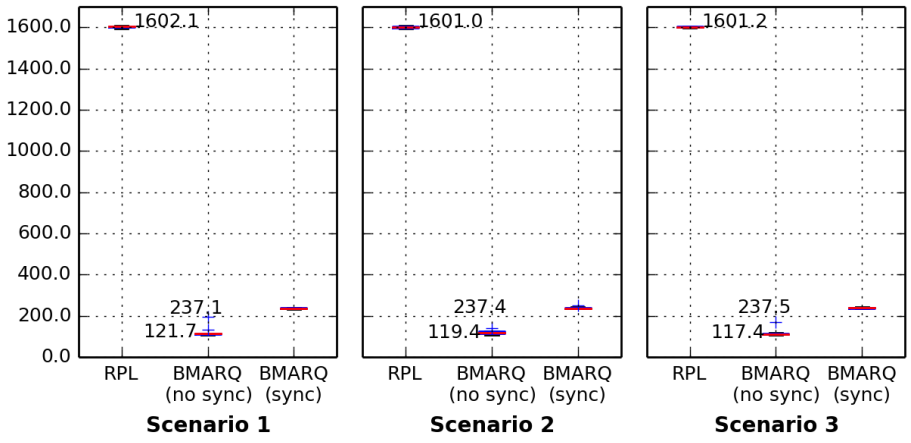

Figure 3.26: Energy consumed by each solution in each scenario (in J)

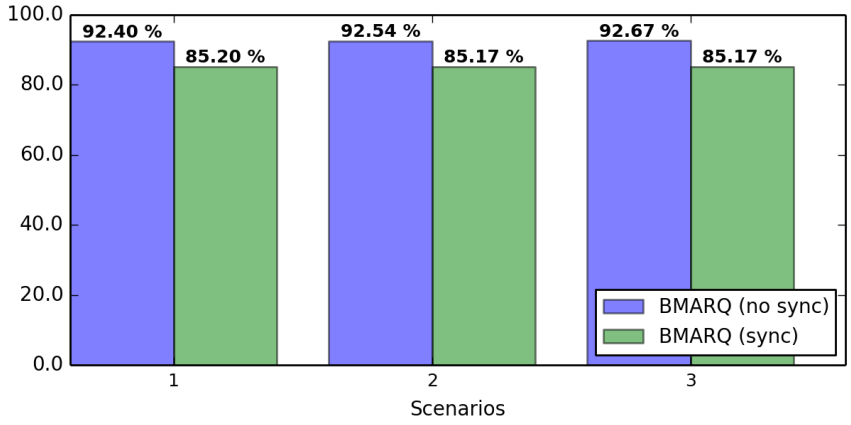

Figure 3.27: RPL-BMARQ energy gain in each scenario (in %)

number of reply packets received by a sink node in response to a query packet, and the number of replies the sink expects to receive (see Eq. 3.7).

$$QSR = \frac{number\_of\_received\_replies}{number\_of\_expected\_replies}, \quad 0 \leq QSR \leq 100\% \tag{3.7}$$

Adapting Eq. 3.7 to our case, the number of received replies is 7. Fig. 3.28 presents the results obtained. One can conclude that RPL presents better result than *RPL-BMARQ*, although the values from both solutions are very close. Please note that in Fig. 3.28 we are representing *QSR* values between 96 and 100%. We can conclude that with *RPL-BMARQ*, QSR does not suffer much, presenting a average value of 98.5%, when compared to *RPL*, which presents a similar value of 99.5%. At the end, the difference value of 1% has no great significance.

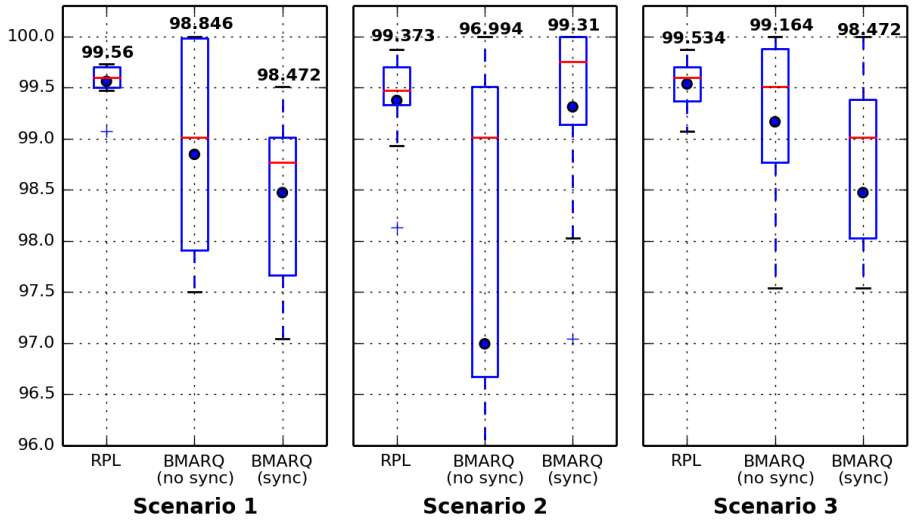

Figure 3.28: Query Success Ratio - QSR (in %)

- **QSR fairness**: fairness metrics are used in network engineering to determine whether users or applications are receiving a fair share of system resources. There are several definitions of fairness. ***Jain's fairness index*** is an example, and it is defined in Eq. 3.8 [115], and it describes the fairness of a set of values where there are n users and x_i is the throughput for the i^{th} user. A straightforward computation shows that the fairness measure γ ranges from $\frac{1}{n}$ (maximum unfairness) to 1 (all x_i are equal) [116]. In the evaluation of our proposed solution also we want to know if the sensor nodes nodes have the same opportunity to receive and to reply to query packets. We also use the *Jain's fairness index* as it is independent of scale, it applies to any number of sensor nodes, and it is bounded between 0 and 1, where $\gamma = 1$ indicates a totally fair network.

$$\gamma(x_1, x_2, \cdots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} x_i^2} \tag{3.8}$$

Applying Eq. 3.8 to this evaluation, x_i corresponds to the *QSR* per node. Fig. 3.29 shows *QSR fairness* mean values obtained from simulations for the 3 scenarios. In average, those values show that all solutions present fairness indexes above 99%. The same figure also presents *fairness* values for each of the application supported by the network, and their average.

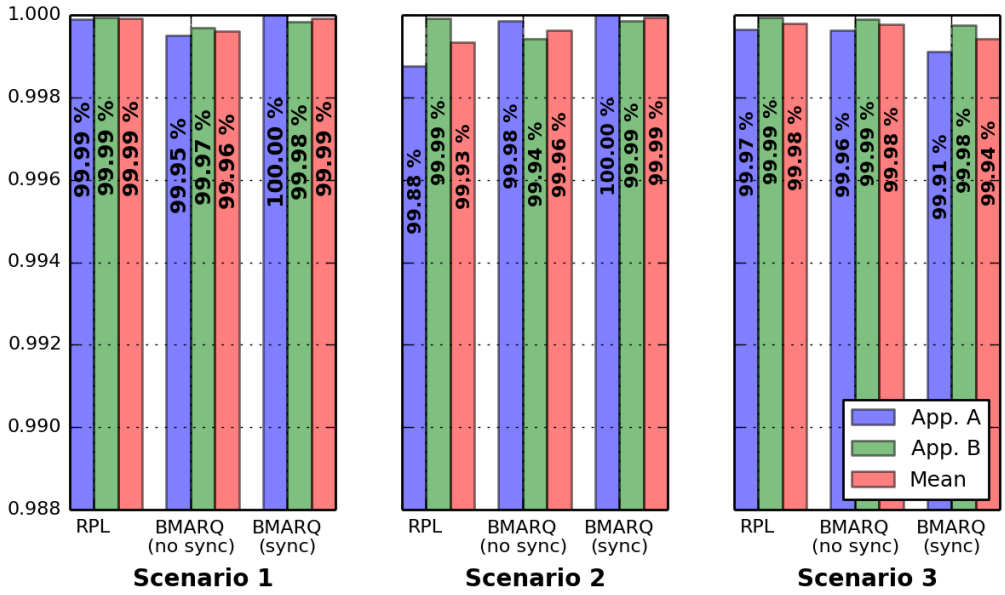

Figure 3.29: QSR fairness

- **Delay**: we define *delay* (Δ) as the time interval between the time instant a query was sent by a sink node and the time the sink receives the correspondent reply, as shown in Fig. 3.30. We compute this delay as

$$\Delta_{k,n} = t_{R_{k,n}} - t_{Q_k} \tag{3.9}$$

where $\Delta_{k,n}$ is the delay for the reply from the node n to a query k, t_{Q_k} is the time the query k was sent, and $t_{R_{k,n}}$ is the time the reply k from the node n was received. Fig. 3.31 presents

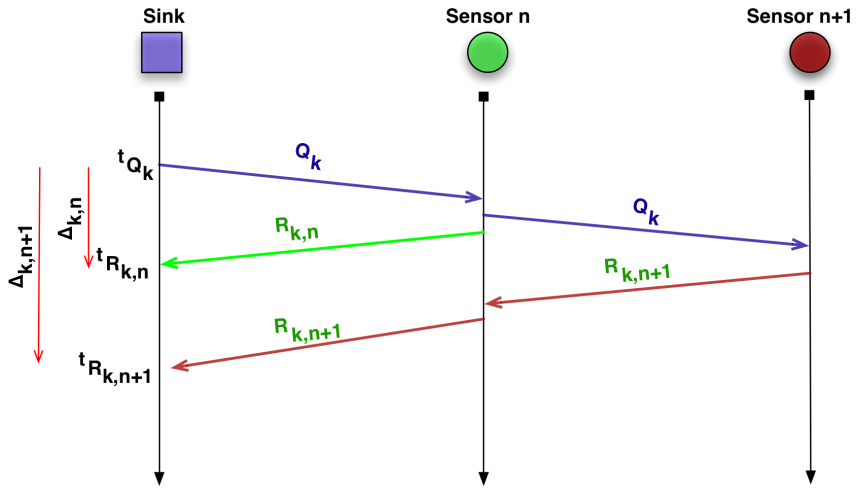

Figure 3.30: Delay definition

the results obtained. These values were computed as defined in Fig. 3.30. We note that with *RPL-BMARQ* we achieve higher mean delays. For instance, in *Scenario 3*, the mean delay value is 1.524 s for *RPL-BMARQ* with the synchronization mechanism implemented, 1.539 s for *RPL-BMARQ* not using the synchronization mechanism, and 1.375 s for *RPL*. This is expected since nodes using *RPL-BMARQ* have bigger processing times. Also, analyzing the DAGs generated (Fig. 3.24) we note that, in average, *RPL-BMARQ* creates longer DAGs so the packets would take more time to reach their destinations. In average, the delay value for *RPL* is 1.24 s, the delay value for *RPL-BMARQ (no sync)* is 1.39 s, and the delay value for *RPL-BMARQ (sync)* is 1.36 s. This corresponds to more $10,8\%$ and $8,8\%$ of delay time, respectively for *RPL-BMARQ (no sync)* and *RPL-BMARQ (sync)* implementations. Finally, analyzing delays for both *RPL-BMARQ* implementations, when nodes use the synchronization mechanism, they would not be active all at same time, but when necessary. This has the effect of reducing communication activities, occupying the transmission media for less time. One problem that our solution may raise is related to load balancing between nodes. There are applications which require the transmission of more packets per time unit than others and, as a consequence, nodes belonging to the subnetwork defined by the high packet rate applications are required to process more packets. This may create load balancing related issues such as higher network delays through some of the network paths. These delays may be higher in our solution than in regular *RPL*. However, if the offered loads are low and stable, as it is the case of the traffic scenarios envisaged, these differences should not be relevant for our claims that are related to energy savings, as may be observed for instance in Fig. 3.31 and in Fig. 3.39, where one application demands the transmission of more packets than the other; therefore, the nodes in the subnetwork defined by the high packet rate application process more packets and delays

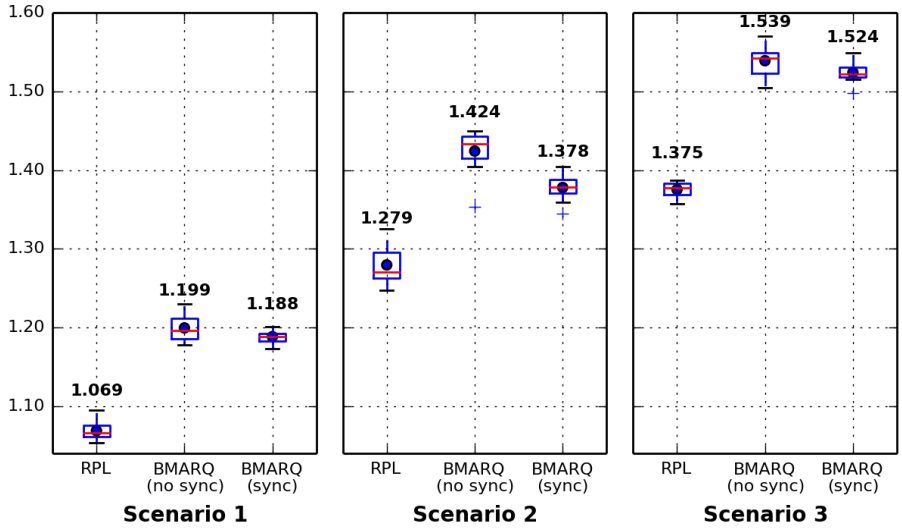

Figure 3.31: Delay (in s)

become higher. *RPL* does not subdivide the network and routing paths do not depend on applications and, for this reason, *RPL* presents lower delays. Fig. 3.31 and Fig. 3.39 enable us to estimate the magnitude of increasing delays (in %) introduced by our solution. Analyzing Fig. 3.31 and comparing to regular *RPL* we can verify that: for scenario 1, and not using the synchronization mechanism, *RPL-BMARQ* presents $10,8\%$ more delay, whereas using the synchronization mechanism the same is 10%; for scenario 2, and not using the synchronization mechanism, *RPL-BMARQ* presents $10,2\%$ more delay, whereas using the synchronization mechanism the increase is $7,2\%$; and for scenario 3, and not using the synchronization mechanism, *RPL-BMARQ* presents $10,7\%$ more delay, whereas using the synchronization mechanism the increase is $9,8\%$. For the case of scenario 4, from Fig. 3.39 we can observe that, when not using the synchronization mechanism, *RPL-BMARQ* presents an increase of delay of $17,7\%$, whereas using the synchronization mechanism the same increase is $16,8\%$. In this scenario delays are higher because there is a node running one application which is required also to process packets from the other application. From the above analysis we may conclude that our solution makes delays to increase about 10%, in average, but our claims on energy still hold.

- **Packets per query**: in the simulations performed, the sink node running *App. A* generates a total of 24 queries, whilst sink node running *App. B* generates 96 queries. Analyzing the data extracted from simulations we could investigate on a *per-query* basis, how many *Layer 3* multicast and unicast packets were sent and received by all the nodes. We consider also routing packets. Fig. 3.32 presents our results. For all the scenarios, both *RPL-BMARQ* solutions present lower mean number of total *Layer 3* multicast packets sent and received. For *Scenario 1*, using *RPL-BMARQ (no sync)*, *per-query*, the mean number of total packets sent is 10 and the mean number of total packets received is 25; using *RPL-BMARQ (sync)*, the mean number of total packets sent is 8 and the mean number of total packets received

is 17; using *RPL*, the mean number is higher (15 packets sent and 45 packets received). For *Scenario 2*, using *RPL-BMARQ (no sync)* the mean number of total multicast packets sent is 9 and the mean number of total multicast packets received is 24; using *RPL-BMARQ (sync)* the mean number of total multicast packets sent is 7 and the mean number of total multicast packets received is 17; using *RPL*, the mean number of total multicast packets is also higher (15 for packets sent and 45 for packets received). Finally, for *Scenario 3*, using *RPL-BMARQ (no sync)* the mean number of total multicast packets sent is 10 and the mean number of total multicast packets received is 26; using *RPL-BMARQ (sync)* the mean number of total multicast packets sent is 7 and the mean number of total multicast packets received is 16; using *RPL*, the mean number of total multicast packets is also higher (15 packets sent and 45 packets received). For the mean number of total *Layer 3* unicast packets sent and received, the results show that also both *RPL-BMARQ* implementation present lower numbers. For *Scenario 1* the mean number of total unicast packets sent using *RPL-BMARQ (no sync)* is 33, using *RPL-BMARQ (sync)* is 29, and using *RPL* is 45; for the same scenario, using *RPL-BMARQ (no sync)* the mean number of total unicast packets received is 77, using *RPL-BMARQ (sync)* is 49, whilst using *RPL* the same is 135. For *Scenario 2* the mean number of total unicast packets sent is 36 for *RPL-BMARQ (no sync)*, 32 for *RPL-BMARQ (sync)*, and 50 for *RPL*. The mean number of total unicast packets received is 76 for *RPL-BMARQ (no sync)*, 61 for *RPL-BMARQ (sync)*, and 141 for *RPL*. Finally, for *Scenario 3* the mean number of total unicast packets sent is 37 for *RPL-BMARQ (no sync)*, 33 for *RPL-BMARQ (sync)*, and 53 for *RPL*, and the mean number of total unicast packets received is 59 for *RPL-BMARQ (no sync)*, 53 for *RPL-BMARQ (sync)*, and 144 for *RPL*.

From above analysis we conclude that the major gain of the *RPL-BMARQ* solution relies on the total number both multicast and unicast packets sent and received which is lower than the equivalent *RPL*. Table 3.5 summarizes simulation results showing the total number of multicast and unicast packets transmitted and received by the nodes in each network solution.

- **Reaction to topology changes**: *RPL-BMARQ* solution behaves just like *RPL* with respect to convergence times, and the reaction to network topology changes are similar to those observed for *RPL*. *RPL-BMARQ* uses the same *Trickle timer mechanism*. When a node does not agree with its neighbors, that node communicates quickly to resolve the inconsistency. On the other and, when nodes agree they slow their communication rate exponentially, and end by exchanging packets very infrequently. Instead of flooding a network with packets, the algorithm controls the sending rate so that each node hears a small trickle of packets, just enough to stay consistent [43].

3.2.3.2 Special Case: Scenario 4

This scenario was simulated as the others, and while running and analyzing preliminary results, this scenario presented an issue: node 9 (sink from application B) does not receive replies from

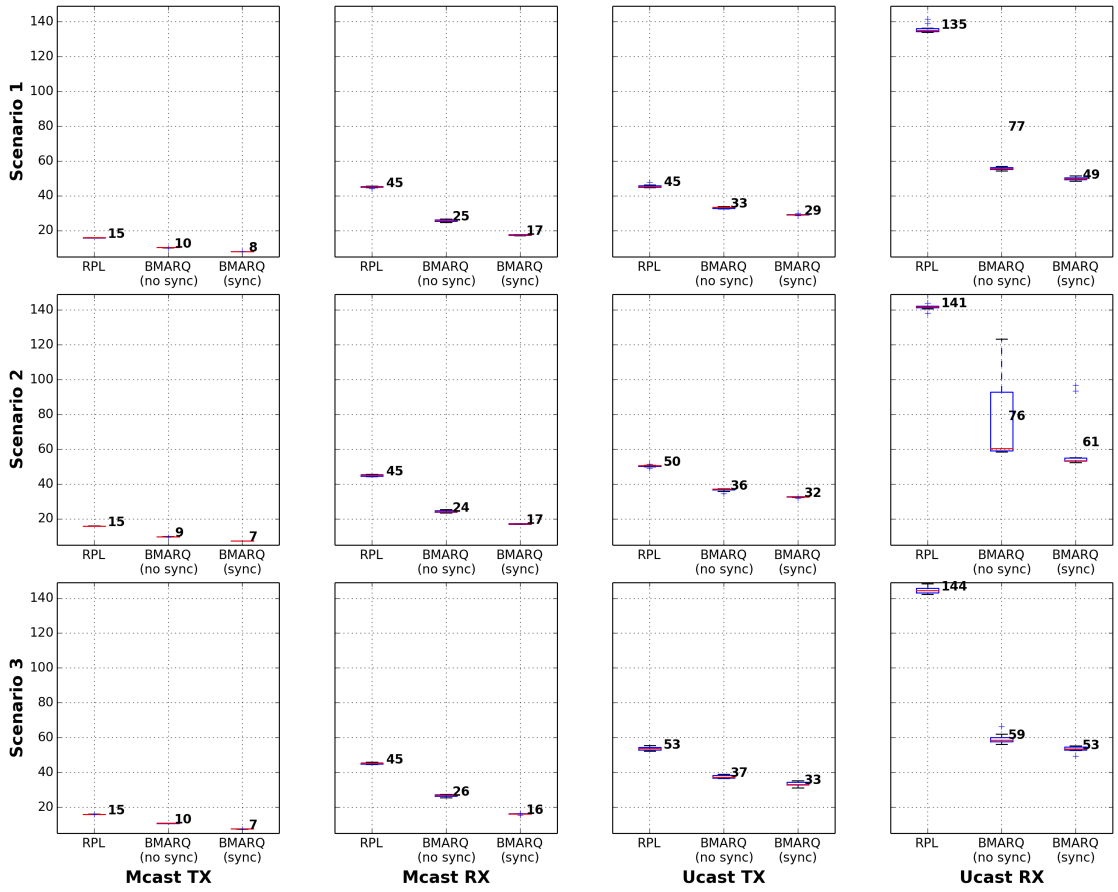

Figure 3.32: Total number of packets per query

Table 3.5: Mean total number of packets per query

Scenario	Solution	Multicast		Unicast	
		TX	RX	TX	RX
1	*RPL*	15	45	45	135
	BMARQ (no sync)	10	25	33	77
	BMARQ (sync)	8	17	29	49
2	*RPL*	15	45	50	141
	BMARQ (no sync)	9	24	36	76
	BMARQ (sync)	7	17	32	69
3	*RPL*	15	45	53	144
	BMARQ (no sync)	10	26	37	59
	BMARQ (sync)	7	6	33	53

nodes 10, 13 and 16, although queries are received and replied by them. This behavior is caused by node 8 which, although running application A, is also parent of node 16 and therefore responsible to forward packets from nodes 16, 13 and 10. Since node 8 selects node 7 as parent (see Fig. 3.24 d.), reply packets are never forwarded by node 7 because it does not run the application of nodes 10, 13 and 16. To solve this issue, node 8 upon the reception of nodes 10, 13 and 16 reply packets, should forward them directly not to node 7, but to node 15, which runs the same application, to allow node 9 to received the expected reply packets. In this kind of scenarios, every node being parent of nodes not running the same application, and having selected as parent a node running the same application, should send directly all received reply packets to a neighbor node running the same application from which reply packets where originated. A close snapshot of this is showed in Fig. 3.33 where node 8 has selected node 7 (fe80::212:7407:7:707) as parent, but detecting that a reply packet is, as example, from node 10 and node 16 (aaaa::212:740a:a:a0a and aaaa::212:7410:10:1010) from the application it doesn't run, changes the link to node 15 (fe80::212:740f:f:f0f).

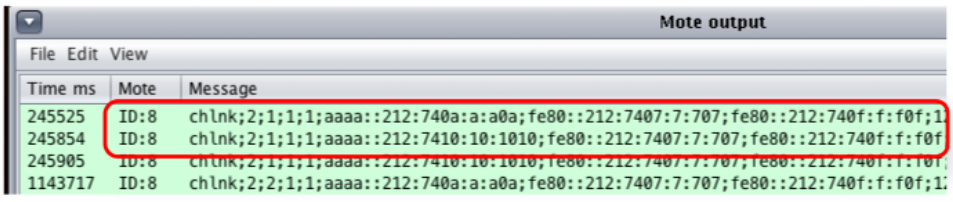

Figure 3.33: Changing link-local address to correctly forward reply packets

- **Energy consumption**: we consider energy consumption related only to communications aspects and from the results obtained we present the durations of relevant communications states in Fig. 3.34. As can be observed, the permanence time in energy consuming states is lower for *RPL-BMARQ* (both *no sync* and *sync* implementations) than for *RPL*.

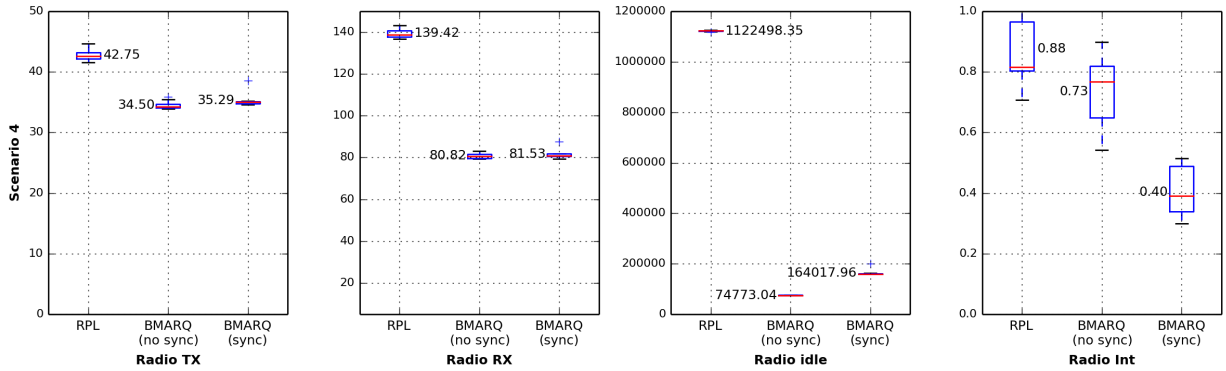

Figure 3.34: Mean total radio activity time for scenario 4 (in s)

We have computed the total energy consumed by the nodes using Eq. 3.3, and presented results in Fig. 3.35. They show that the total energy consumed by the nodes running *RPL-BMARQ* is very low, as it turns the nodes radio *off* during more time than *RPL*. We have also computed the energy gain for *RPL-BMARQ* as described in Sec. 3.2.2.2. Results show that, when the synchronization mechanism is not used the nodes spend about 93% less energy than the same nodes running standard *RPL*. On the other hand, if the synchronization mechanism is used, the nodes still spend less energy (about 85%). Using the *RPL-BMARQ*

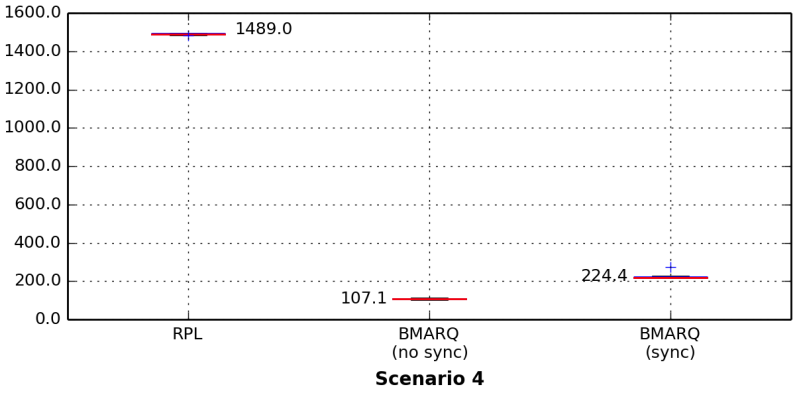

Figure 3.35: Energy consumed by each solution for scenario 4 (in J)

solution not implementing the synchronization mechanism, the nodes spend about 93% less energy than the same nodes running *RPL*. In the case of the second implementation of *RPL-BMARQ* in which the synchronization mechanism is used, the nodes spend about 85% less of energy than *RPL*, as shown in Fig. 3.36.

- **Query Success Ratio**: Fig. 3.37 shows simulation results for *Query Success Ratios*. Although the values from both solutions are very close, *RPL* presents better results than

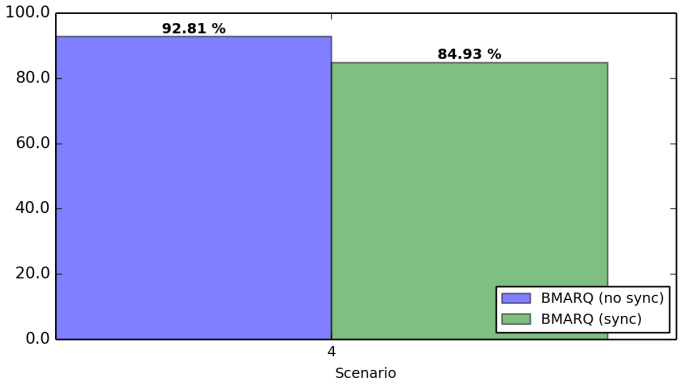

Figure 3.36: RPL-BMARQ energy gain for scenario 4 (in %)

RPL-BMARQ. Please note that in this figure we are representing *QSR* values between 96 and 100%, and considering that the difference value is about 1%, we can conclude that with *RPL-BMARQ* QSR does not suffer much.

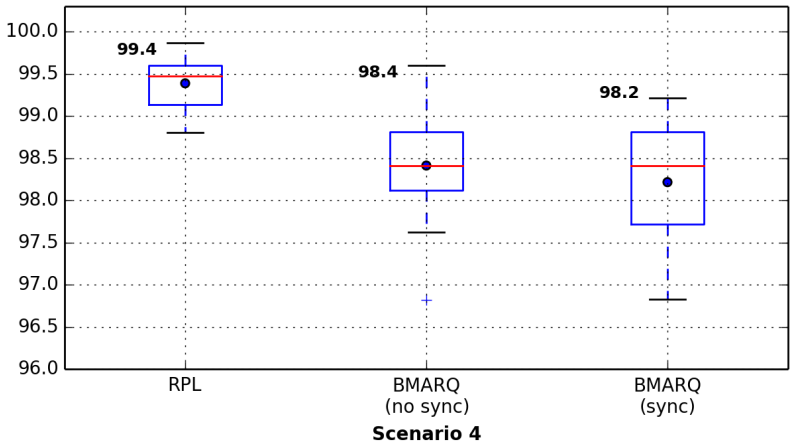

Figure 3.37: Query Success Ratio (QSR) for scenario 4 (in %)

- **QSR fairness**: Fig. 3.38 shows that *QSR fairness* values obtained from simulations present similar fairness indexes (about 99.9%) for both solutions.

- **Delay**: Fig. 3.39 shows delay values using both solutions. We note that with *RPL-BMARQ* we achieve higher delays. The mean delay value for *RPL-BMARQ* is 1.55 s, and 1.28 s for *RPL*. This is also expected since in this scenario *RPL-BMARQ* constructs a longer DAG (see Fig. 3.24), so nodes use more hops to communicate. Also, in each node is used more processing time.

- **Packets per query**: In the simulations performed, the sink node running *App. A* generates a total of 24 queries, whilst sink node running *App. B* generates 96 queries. Analyzing the data extracted from simulations we evaluated on a *per-query* basis how many *Layer 3*

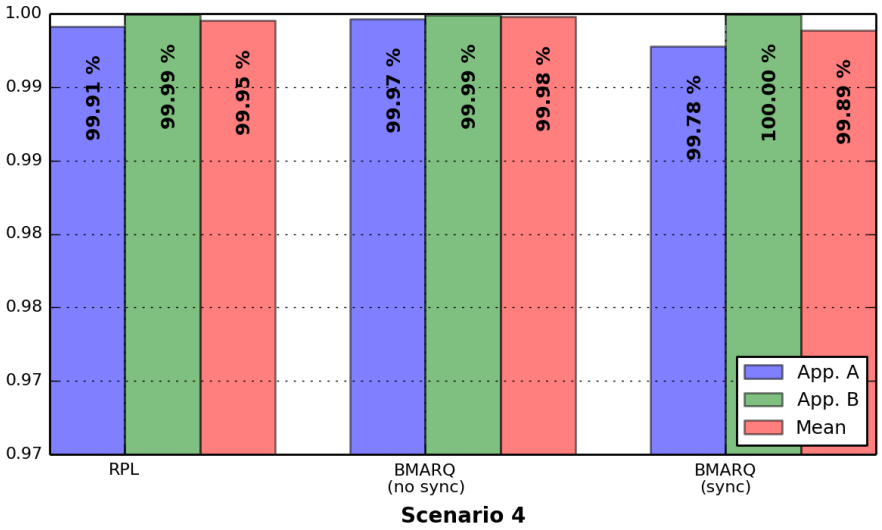

Figure 3.38: QSR fairness for scenario 4 (in %)

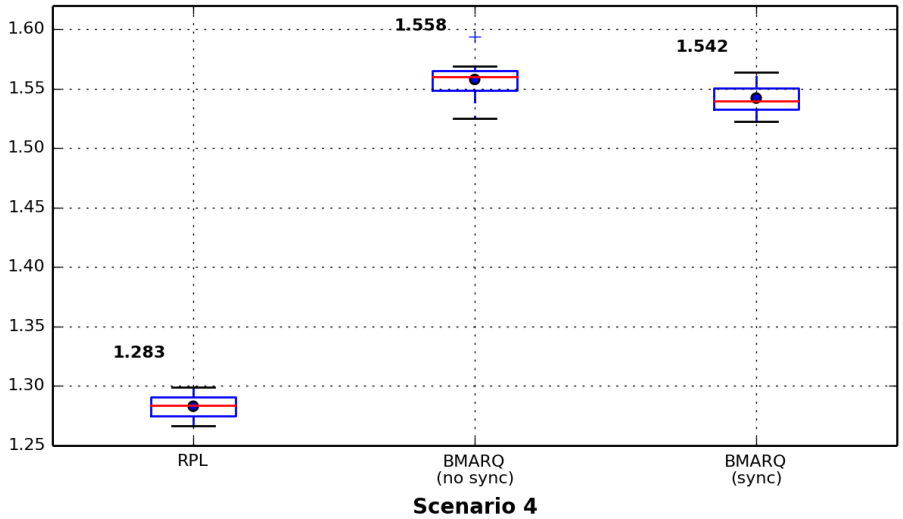

Figure 3.39: Delay for scenario 4 (in s)

multicast and unicast packets were sent and received by all the nodes. We consider also routing packets.

Fig. 3.40 shows on a *per-query* basis, the number of packets transmitted and received. Both *RPL-BMARQ* solutions present lower mean number of total *Layer 3* multicast packets, sent and received. Using *RPL-BMARQ (no sync)*, the mean number of total packets sent is 11 and the mean number of total packets received is 29; using *RPL-BMARQ (sync)*, the mean number of total packets sent is 9 and the mean number of total packets received is 22; using *RPL*, the mean number is higher (15 packets sent and 45 packets received). For the mean number of total *Layer 3* unicast packets sent and received, the results show that also both *RPL-BMARQ* implementations present lower numbers. The mean number of total unicast packets sent using *RPL-BMARQ* (in both implementations) is 44, whilst using *RPL* this value is 50; the mean number of total unicast packets received using both implementations of *RPL-BMARQ* is 82 and using *RPL* the same is 135.

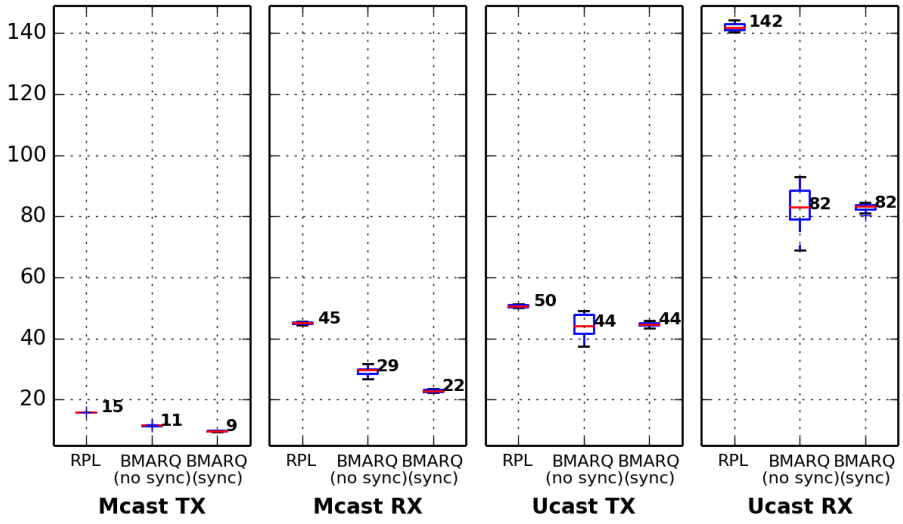

Figure 3.40: Total number of packets per query for scenario 4

From the above analysis we conclude that the *RPL-BMARQ* presents the same behavior as in the other scenarios, and that the major gain introduced by *RPL-BMARQ* solution is the capability to wake and to sleep the sensor nodes in an synchronized manner, reducing radio activity time, while maintaining *QSR* and *fairness* ratios high.

3.2.4 Testbed Experiments

In order to confirm the results obtained from simulations, we also tested *RPL-BMARQ* in a real environment. For that purpose, two of the scenarios studied were selected (scenarios 1 and 3) and deployed. Since it was not possible to reproduce them at same scale, the scenarios deployed corresponds to a 3x3 square lattice topology, while keeping all the assumptions of the evaluated scenarios. In order to obtain reliable terms of comparison, we have simulated these deployments

using the same methods as in section 3.2 and compared the simulated results with those obtained in testbeds. Fig. 3.41 shows both deployments, which were realized in an *Auditorium* inside *Escola Superior de Tecnologia e Gestão de Viseu* facilities (see Fig. 3.42). The nodes were placed at a distance of 5 meters, and the radio transmission power was reduced to -7dBm in order to reduce nodes radio influence space. Application A runs in five nodes (1, 2, 3, 4 and 5), while application B in four nodes (9, 10, 11 and 12). Node 9, is at the same time the root of the DAGs and sink. Node 1 is the other sink.

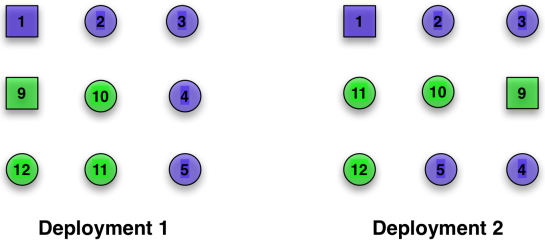

Figure 3.41: Scenarios deployed

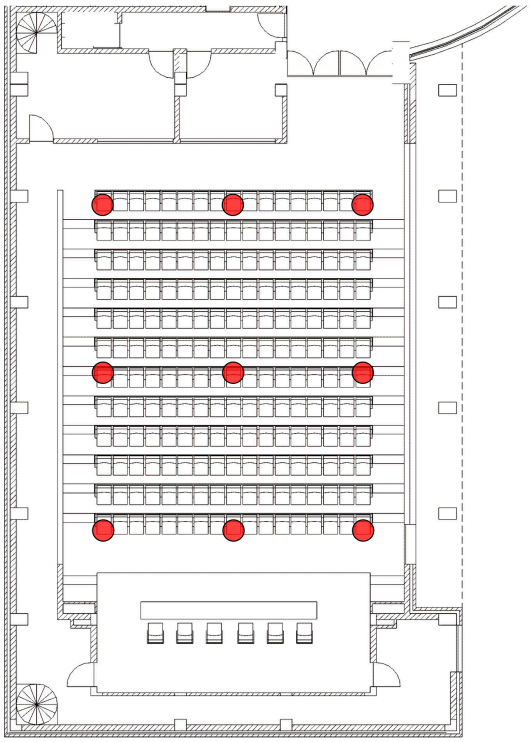

Figure 3.42: Local of testbed deployment

3.2.4.1 Energy Metering Hardware

It was necessary to measure real power consumption to compare with the energy consumption obtained from simulations. Again, a node (node 5) was randomly selected to measure real power

consumption. For this, it was necessary to implement a small *Energy meter*, which could measure in real-time the energy consumed by this node. Fig. 3.43 shows this implementation which uses a *BeagleBone Black* platform [117] responsible to acquire analog values from the measured node. These values are acquired using an instrumentation amplifier which samples the voltage drop of a serial shunt resistor, supplied to the measured node. This platform runs a python program which acquires every second 25 samples, returning their mean value, and uses it to compute the power consumption. For accuracy it was necessary to measure real values, compare and correct them from the nodes datasheet [18], and use them in the simulations to reflect real values. As such, for power supply the value was corrected to 3 V; and the power consumption when MCU is on and the Radio Off to 12 mA.

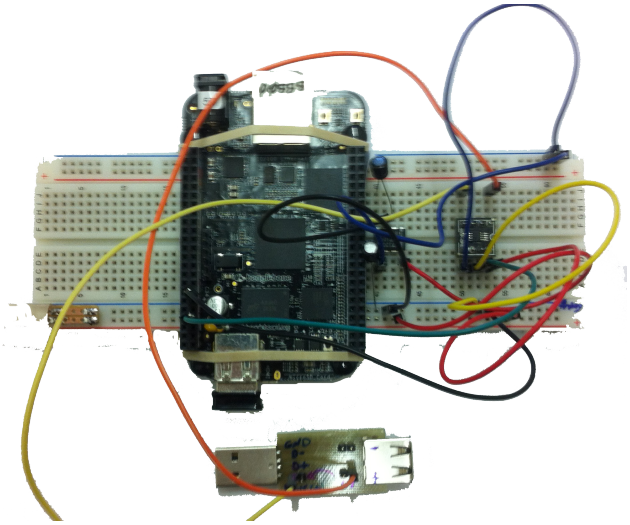

Figure 3.43: Energy meter implemented

3.2.4.2 Results and Discussion

- **Energy consumption**: Fig. 3.44 shows simulation and real implementation results for the scenarios deployed. Simulation and real implementation results for the scenarios deployed demonstrated that, in the real testbed implementations, the node consumes a little more energy than in simulations. Additionally the energy recorded corresponds not only to other hardware components consumption that the nodes have (e.g. LEDs, ADCs) which were not considered in the simulation platform, but also to the total number of transmitted and received packets by the node which was not possible to record. In the first deployment, simulation showed that the node consumes 485.83 J, and in the second 534.26 J. Node real measurements have shown a consumption of 567.73 J and 552.41 J. In the first deployment the node consumes more 16.8% of energy whereas in the second the same node consumes more 3.4% of energy, when compared to simulations results. From these results, and considering that some hardware platform components were not considered in the simulations, we

can conclude that the real node energy consumption in each deployment can be considered as valid.

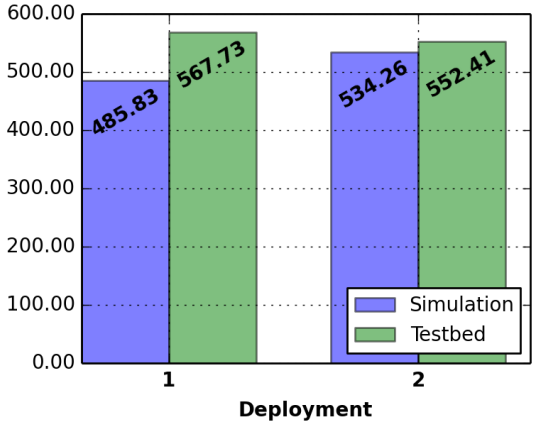

Figure 3.44: Energy consumed by the selected node for each scenario (in J)

- **Query Success Ratio**: Fig. 3.45 shows simulation and real implementation results. As it can be seen, both present same values (100%), which means that also in real testbeds, the nodes reply to all the queries sent by sinks, maintaining their synchronization.

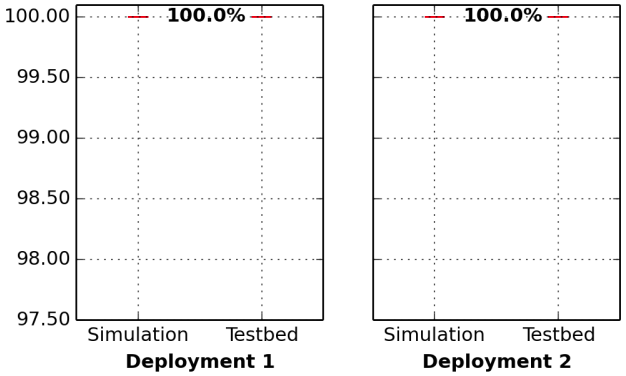

Figure 3.45: Query Success Ratio (QSR) for the scenarios selected (in %)

- **QSR fairness**: Fig. 3.46 shows *QSR fairness* mean values obtained from simulations and testbeds implemented. In average, *QSR fairness* values obtained from simulations and testbeds implemented present same values (100%). This also means that in real testbeds, the nodes have the same opportunity to reply to all the queries sent by the sinks, as all the queries are equally received by all the nodes.

- **Delay**: Fig. 3.47 shows simulation and real implementation delay results. As one can observe, both present almost the same values (about 900 ms), which means that nodes in the testbeds have the same behavior as in the simulation environment. Note that in this figure we are representing *delay* between 600 and 950 ms.

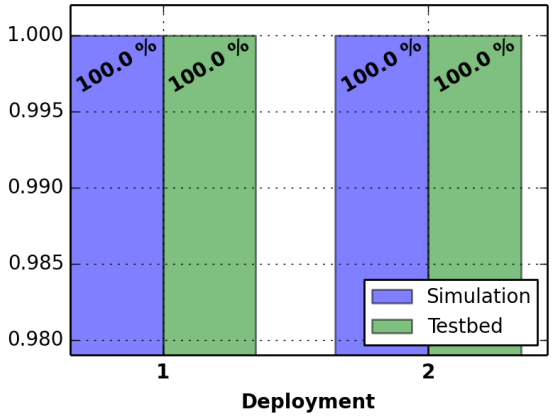

Figure 3.46: QSR fairness for scenario 4 (in %)

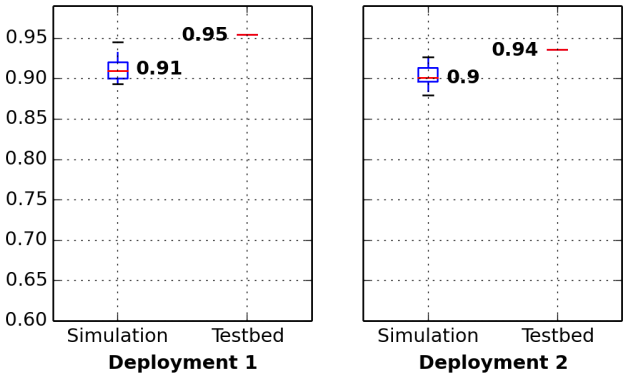

Figure 3.47: Delay for the selected scenarios (in %)

From the above results we can conclude that there are no major differences between what was observed in the simulation environment and that what was expected in the testbed environment, and consider valid the testbeds, confirming the usability of *RPL-BMARQ*.

3.3 Summary

This chapter describes our first contribution. In section 3.1 we present and discuss our *Application-Driven WSN* concept, and describe how we changed the *RPL* routing protocol in order to constrain RPL-defined routing trees. Section 3.1.1 presents the *RPL-BMARQ* communications stack and describes its cross-layer characteristics. Section 3.1.2 describes the mechanism added to *RPL* as an extension (*RPL-BMARQ*) in order to create the *DAG*s according to our rationale.

By the time we were conducting our research, the *ContikiOS* did not fully supported *IPv6 multicast*. Therefore, we specified an *Application-Driven multicast* mechanism, which is described in Section 3.1.3.

In Section 3.2 we evaluated the proposed solution, first by conducting theoretical studies to estimate the gains introduced, second by simulating the solution, and third by implementing two testbeds, being the results presented and discussed.

Theoretical studies showed that when using *RPL-BMARQ* the sensor nodes send and receive less packets, as they are kept as much as possible in sleep state, waking only when necessary (according to their applications duty cycle). As a result, the solution presents less energy consumption.

The results obtained from the simulations confirmed the theoretical results, and showed that the sensor nodes nodes have the same opportunity to receive and to reply to query packets, presenting, in average, a *fairness* of 99%. Also, simulations showed that most of the queries sent by the sink nodes are replied, which is reflect by a *QSR* of about 99.5%. On the other hand, our solution makes delays to increase about 10%, in average, but our claims on energy still hold; compared to *standard RPL*, our solution presents, in average, energy gains of about 92%, which enables us to conclude that *RPL-BMARQ* extends the network lifetime.

Finally, in what concerns the testbeds, it was necessary to implement a *energy metering hardware* in order to measure real power consumption to compare with the energy consumption resulted from simulations. The obtained results from the testbeds demonstrated that the sensor nodes consume a little more energy than in simulations because the energy recorded includes hardware components which were not considered in the simulation. The results showed that *QSR* and *fairness* values are the same (100%). From these results we can conclude that there are no major differences between what was observed in the simulation environment and that what was expected in the testbed environment, confirming the usability of *RPL-BMARQ*.

Chapter 4

Application-Driven WSN Node Synchronization for *RPL-BMARQ*

In Chapter 3 we defined *Application-Driven WSN* as a cross-layer solution aimed to help reducing the energy consumed by a network of sensors executing a set of applications. This paradigm assumes that each application defines its own network and set of nodes so that the exchange of information can be confined to the nodes associated to the application. The nodes share information about the applications they run and their duty-cycles, and nodes are put asleep when there is no activity related to their applications. When nodes receive a query packet they know exactly when they must wake up on the next period. The nodes alternate between wake and sleep states, and the amount of time spent in each phase is determined by the applications duty cycle. When the wake up time expires, the node switches to the sleep state, waking up again by the time computed by the synchronization mechanism proposed in this thesis.

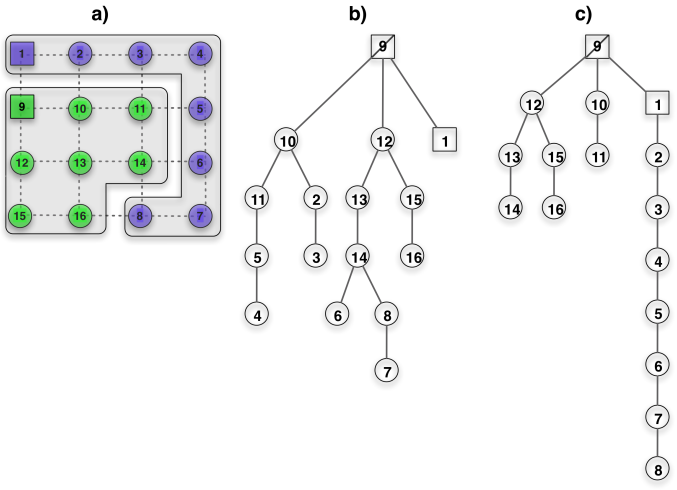

Figure 4.1: Application-Driven WSN concept. a) network topology; b) RPL DAG; c) BMARQ-RPL DAG

We assume that every node can participate in route discovery and packet forwarding. However, the nodes forwarding a given type of data will be primarily selected from the set of nodes running the same application to which the data is associated. For that purpose, each query packet includes information about the associated application (*APPID*), which is known by the nodes running that application. Our routing scheme tries to insure that data of an application is relayed mainly by the nodes running that application. When the sink node queries the other nodes running the same application, routing paths follow the *Directed Acyclic Graph* (DAG) created. This DAG is created and maintained by a change to the RPL protocol scheme which uses mainly the nodes running that application; the nodes not associated to this application will not participate in routing process, in a first attempt. In our proposal the subset of nodes running the same application forms a "subnetwork" with multi-hop connectivity and application packets carry out also information about the applications duty cycle (T_{CYCLE} and T_{ON}) that is used to create and maintain the *DAGs* in which not only the the nodes running the same application, but also the nodes having the same application duty cycle can be "grouped". Fig. 4.1 a) shows a network topology supporting two different applications. Fig. 4.1 b) shows the DAG created with standard RPL, and Fig. 4.1 c) shows the DAG created by our proposed solution. The wake up mechanism is based on the applications time cycle information (*TCYCLE* and *TON*), carried by every application query sent by the sink nodes. When a node receives a query packet it knows exactly when it must wake up on the next period.

4.1 Application-Driven Synchronization Mechanism

According to our application-driven concept, synchronization is achieved between the nodes that run the same applications, or between the nodes that have the same application duty-cycle, by considering their duty-cycles. Therefore, the first time a node joins the network it waits for an application query packet to adjust its virtual clock to the time carried by the query packet. We realize that this corresponds to setting the time's nodes to a value which does not consider network delays but, as demonstrated below, this has no impact on our synchronization mechanism as the nodes dynamically adjust their sleeping offset (see $\beta \cdot |\delta_{k,n}|$ component in Eq. 4.2) and wake-up and sleep almost at same time during the network lifetime. As such, the synchronization algorithm takes advantage of the application query packets that are sent by the sink nodes once in every application duty-cycle to maintain the sensor nodes synchronized. A network may support several applications but only the nodes running the same application, or having the same duty-cycle will synchronize between them. Therefore, a network supporting different applications may have different sets of nodes with different synchronizations, and still be fully functional. Without having to send or to receive other type of packets for synchronization purposes, the nodes will rely only on the queries received to synchronize. In fact, this algorithm is centralized on a sink node but its design is simple and adequate for our purposes. A distributed design would be more

complex and imply the use of other type of packets for synchronization, often broadcasted through the network, which would have impact in energy consumption due to packet transmission and reception costs.

It is unlikely that all the sensor nodes would join a network at the same time. Having the nodes active during all the time would deplete their batteries, so nodes have to go sleep and to wake up periodically. All the nodes have to be awake almost at same times in order to receive sink queries and to forward them to the other nodes. As a result, nodes must be synchronized according to the application cycle they run. In order to synchronize all the nodes in the network, our proposed synchronization mechanism uses a synchronous method which includes two phases: *the synchronization setup phase* and *the synchronization maintenance phase*, described below.

4.1.1 The Synchronization Setup Phase

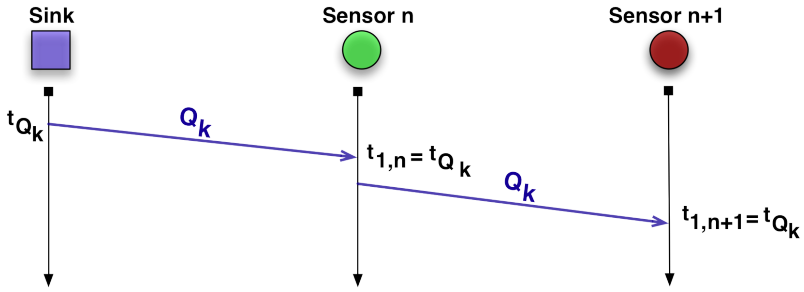

Figure 4.2: Synchronization setup phase

When a sensor node joins the network, it remains in the wake state and waits for the reception of its first query packet sent by the sink node and forwarded by other nodes. Upon its reception, the node adjusts a virtual clock to the timestamp carried by the query. As it can be observed from Fig. 4.2, the query packet sent by a sensor *node n* towards a sensor *node n+1* is the same query packet that *node n* received from the sink node. The timestamp carried by the query is extracted from the query packet. This phase is used to readjust the virtual clock; the periodicity of this readjustment depends on how often the nodes have to readjust their virtual clock. It is known that this phase corresponds to setting the time's nodes to a value which does not consider network delays.

In the example shown in Fig. 4.3 a), sink node A issues a query ($Q_{k,j}$) before sink node B. The query packet is disseminated through the network as expected using the *RPL-BMARQ* routing solution [14]. Sensor nodes C and D, which run this sink's application, set their virtual clock to the timestamp carried out by the packet. Sensor node E, not running this application, also sets his virtual clock to the timestamp carried out by the query packet since it is the first query it receives. The same query packet ($Q_{k,j}$) is then forwarded to the other sensor nodes (nodes G, H, and K) which will also set their virtual clock to the same timestamp. Sensor node E will not forward the query packet $Q_{k,j}$ since it does not run this application, and does not have neighbors running it. Similarly, node F, upon the first query packet ($Q_{k,i}$) reception from sink node B, and because it runs the same application, adjusts it virtual clock to the time carried out by the sink B query

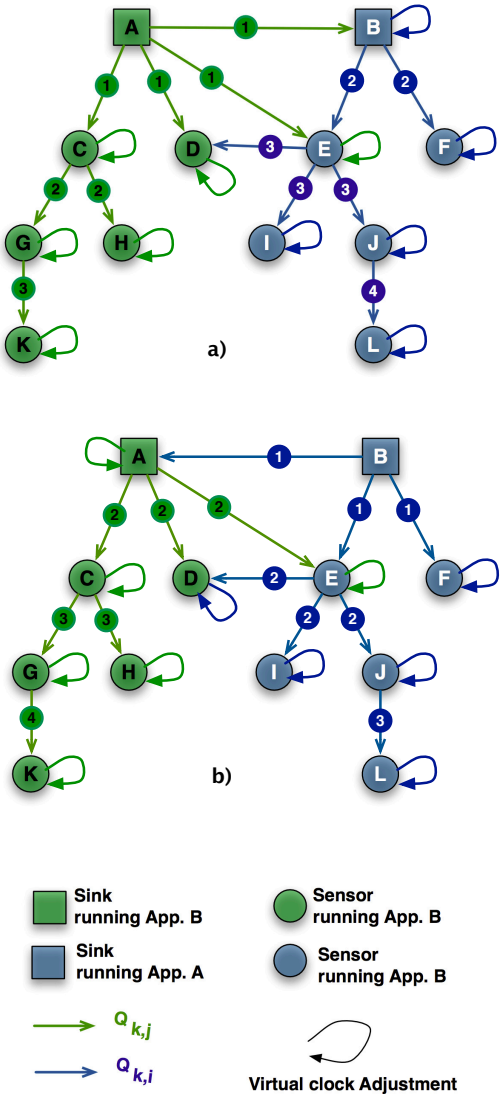

Figure 4.3: Example of nodes synchronization

packet. As this sink has already adjusted is virtual clock using the sink A timestamp, sensor node F will have the same time as the other nodes. Again, the query $Q_{k,i}$ will be forwarded to the other sensor nodes (nodes I, J, and L) which will perform the same virtual clock adjustment. Fig. 4.3 b) shows the same virtual clock adjustments, but in this case it is sink node B that issues the first query packet and adjusts all the network node's virtual clocks.

4.1.2 The Synchronization Maintenance Phase

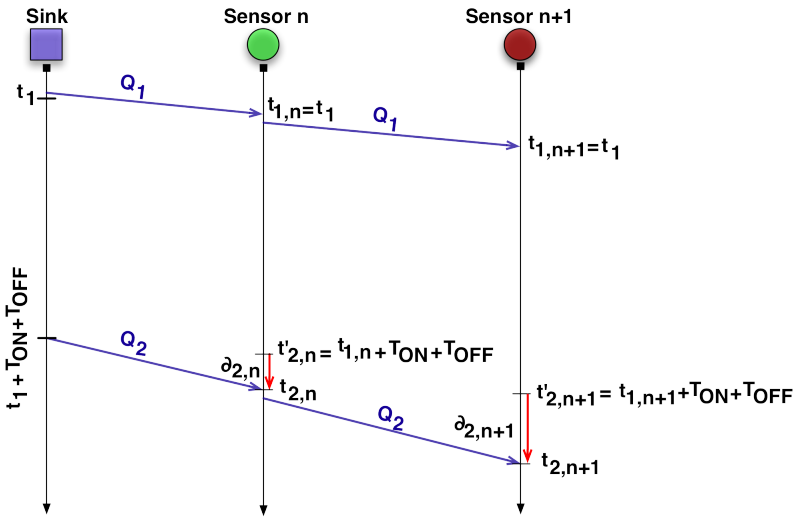

Figure 4.4: Synchronization maintenance phase

Since all the nodes know the characteristics of the applications they run, after the reception of the first query packet they expect to receive the second query packet by $t'_2 = t_1 + T_{ON} + T_{OFF}$. The time the nodes are sleeping (T_{OFF}) is defined as $T_{Cycle} - T_{ON}$ where T_{Cycle} is the application duty cycle time and T_{ON} is the time the nodes are awaked during each duty cycle. However, because network delays are variable, the nodes will receive this second query packet not in t'_2 but in t_2, as shown in Fig. 4.4. There is a difference between the expected value t'_2 and the real value t_2, $\delta_2 = t'_2 - t_2$. For example, if a node is expected to receive a query packet by $t'_2 = 100$ and receives it by $t_2 = 102$, then $\delta_2 = -2$. A negative value means that a query was received in delay, and a positive value means that the query was received in advance. Moreover, delays are the sum of all per-hop delays for each sensor query packet reception and characterized by the sum of the processing and queueing delays in intermediate and destination sensor nodes, and the transmission delays and propagation delays in intermediate nodes. An in depth characterization of these delays may be found in [118].

Our proposed mechanism estimates $\delta_{k,n}$ by using the *Exponentially Weighted Moving Average* (EWMA) technique (see Appendix A). According to Fig. 4.4, the difference between the expected time to receive the next query and the time it is really received is computed by Eq. 4.1

$$\begin{aligned} t'_{k,n} &= t_{k-1,n} + TON + TOFF \\ \delta_{k,n} &= (1-\alpha)\cdot\delta_{k-1,n} + \alpha\cdot(t'_{k,n} - t_{k,n}); 0 < \alpha < 1 \end{aligned} \tag{4.1}$$

where $t'_{k,n}$ is the expected packet reception time, and $t_{k,n}$ is the real packet reception time. $\delta_{k,n}$ is evaluated according to EWMA as in Eq. A.1 with α reflecting the weight of last observation. The $\delta_{k,n}$ value is dynamically adjusted every time a node wakes and receives a query packet, and it is used to control the time the node would sleep in the next cycle, given by Eq. 4.2.

$$T_{Sleep_{k,n}} = T_{OFF} - \beta \cdot |\delta_{k,n}| \tag{4.2}$$

In Eq. 4.2, the β factor is used to amplify the $\delta_{k,n}$ value to guarantee that the sensor node will wake sometime before the next application cycle. $\beta \cdot |\delta_{k,n}|$ is the sleeping offset and represents the time the node will wakeup before the start of the next application duty cycle. Algorithm 3 shows the pseudo-code of the *Application-Driven Synchronization Mechanism* with values given to α and β, and to the virtual clock adjustment periodicity time (*adjust_periodicity_time*).

Algorithm 3: *Pseudocode of the proposed synchronization mechanism*

foreach *(app.query$_k$ received)* **do**
- $\alpha = 0.125$;
- $\beta = 10$;
- **if** *(first(app.query)* **then**
  - set_clock(query$\rightarrow T_{TX}$);
  - adjust_periodicity_time = 3600 · 24 (eg. 24 hours);
  - adjust_counter = $\frac{adjust\_periodicity\_time}{T_{CYCLE}}$;
- **else**
  - **if** *(app.query_id == node.app_id)* **then**
    - t'$_{k,n}$= t$_{k-1,n}$ + T$_{ON}$ + T$_{OFF}$;
    - t$_{k,n}$ = node.query$_{T_{RX}}$;
    - $\delta_{k,n} = (1-\alpha) \cdot \delta_{k-1,n} + \alpha \cdot (t'_{k,n} - t_{k,n})$;
    - $T_{Sleep_{k,n}}$ = app.T_{OFF} - $\beta \cdot |\delta_{k,n}|$;
    - adjust_sleep_timer($T_{Sleep_{k,n}}$);
    - adjust_counter = adjust_counter - 1;
  - **end**
  - **if** *(adjust_counter == 0)* **then**
    - set_clock(query$\rightarrow T_{TX}$);
    - adjust_counter = $\frac{adjust\_periodicity\_time}{T_{CYCLE}}$;
  - **end**
- **end**

end

4.2 Evaluation

In order to validate this mechanism, first we present a study on how the nodes can maintain their synchronization by estimating and evaluating the parameters presented in Eq. 4.1 and Eq. 4.2, which corresponds to investigate in depth the *synchronization maintenance phase*. We also present and discuss results from the proposed synchronization mechanism using different values for α and β parameters, and *Query Success Ratio* (QSR) results from simulations performed in [14], and finally present and discuss some of the results obtained from two real testbeds. The QSR metric [14] is defined as the ratio between the number of reply packets received by a sink node in response to a query packet, and the number of replies the sink expects to receive.

4.2.1 Theoretical

The nodes synchronization mechanism was evaluated considering the following probabilistic distribution of network delays: 1) constant delay; 2) uniform distribution; 3) gaussian distribution; and 4) exponential distribution.

Fig. 4.5 shows one sink node and three sensor nodes. The sink node transmits queries regularly. Each query time reception is affected by those different network delays, and the sensor nodes upon their reception will adjust their sleep time in order to try to wake up at same time on the next application duty cycle. For each node different mean delays were considered: sensor node 1: 0.5 s; sensor node 2: 1 s; sensor node 3: 2 s.

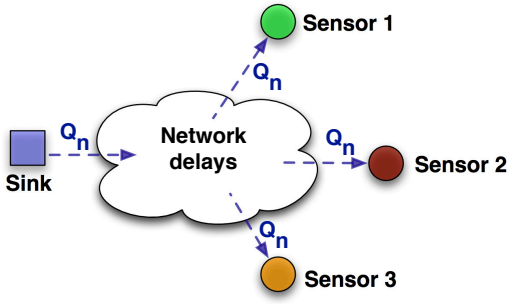

Figure 4.5: WSN delay model

A Python program was written in order to randomly generate different network delay distributions. The program generates 10^5 queries, uses Eq. 4.1 to estimate the new expected query reception time by each node, and uses it to adjust the time each node must sleep (Eq. 4.2) in order to wake up on time for the next application cycle. Finally, the program computes how many time the nodes are waked up simultaneously. We consider that nodes are simultaneously awaked up if the 3 sensors are awaked for at least $\Delta = 80\% \cdot T_{ON}$. Let us also define $T_{Sensors_{ON}}$ as a random variable which captures the time during which the 3 sensors are simultaneously on the *ON* state, having values $T_{Sensors_{ON}} \in [0s, T_{ON}s]$ (see Fig. 4.6). An occurrence of $T_{Sensors_{ON}}$ is computed as the time the first sensor goes asleep minus the time the last sensor wakes up.

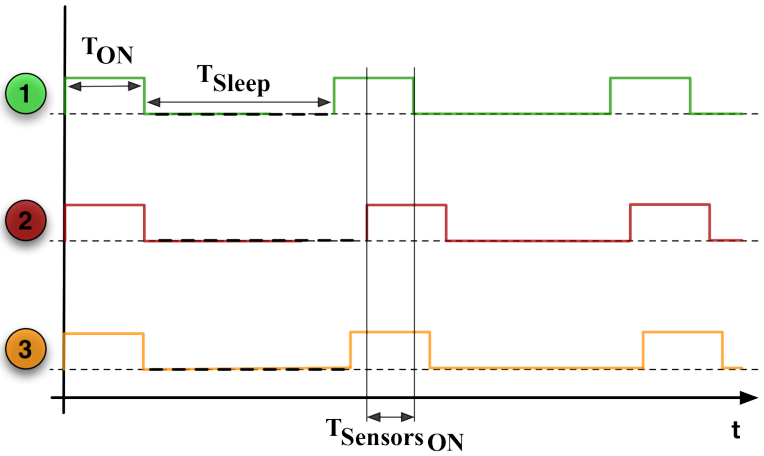

Figure 4.6: Nodes adjustment of T_{ON} simultaneity

In a first attempt, for α in Eq. 4.1 the value was set to 0.125, following current IETF recommendations for managing TCP timers [119], and for Eq. 4.2, the β value was empirically set to 10. All the sensor nodes wake every 15 min remaining waked for 1 min (T_{ON}=60 s and T_{OFF}=840 s).

Fig. 4.7 shows results from the first situation evaluated - constant network delays. In Fig. 4.7 a) we can see the histogram of these delays: 0.5 s for *sensor 1*; 1.0 s for *sensor 2*; and 2.0 s for *sensor 3*. In Fig. 4.7 b) is shown $T_{Sensors_{ON}}$'s histogram; $T_{Sensors_{ON}}$ has always the same value, which equals almost T_{ON} (60 s); we observe that the probability $P[T_{Sensors_{ON}} \geq \Delta] = 1$, which means that the nodes will always be synchronized. We can infer from Eq. 4.2 that $\delta_{k,n}$ has always a value of zero, what also demonstrates the validity of our Python program.

Fig. 4.8 shows results for the second situation evaluated - uniformly distributed network delays, with delays varying between $\pm 20\% \cdot 0.5$ s, $\pm 20\% \cdot 1.0$ s, and $\pm 20\% \cdot 2.0$ s. In Fig. 4.8 a) one can see the histogram of randomly generated delays; Fig. 4.8 b) shows $T_{Sensors_{ON}}$'s histogram. Again we can observe that $P[T_{Sensors_{ON}} \geq \Delta] = 1$. In fact, it is verified that $T_{Sensors_{ON}} \in [57.88, 59.66]$ s, and the mean value of $E[T_{Sensors_{ON}}] = 58.7$ s. As in the first situation, the nodes maintain synchronism in all the cycles.

Fig. 4.9 shows results for the third situation evaluated - gaussian distributed network delays, with delays having a standard deviation which is 20% of the mean values which are 0.5 s, 1.0 s and 2.0 s respectively. In Fig. 4.9 a) we can observe the histogram of randomly generated delays; Fig. 4.9 b) shows $T_{Sensors_{ON}}$'s histogram. As it can be observed, $\delta_{k,n}$ factor from Eq. 4.2 also affects the time each node must sleep ($T_{Sleep_{k,n}}$). Similarly to the previous cases $P[T_{Sensors_{ON}} \geq \Delta] = 1$, and the mean value is $E[T_{Sensors_{ON}}] = 58,77$ s. In this situation the nodes will also maintain synchronism in every application cycle.

Fig. 4.10 shows results from the last situation evaluated - exponentially distributed network delays, with mean delays targeting 0.5 s, 1.0 s and 2.0 s respectively. In Fig 4.10 a) is shown the histogram and, as expected, there are variations; Fig. 4.10 b) shows $T_{Sensors_{ON}}$'s histogram. As can be observed, there are situations where the success condition is not satisfied. In this

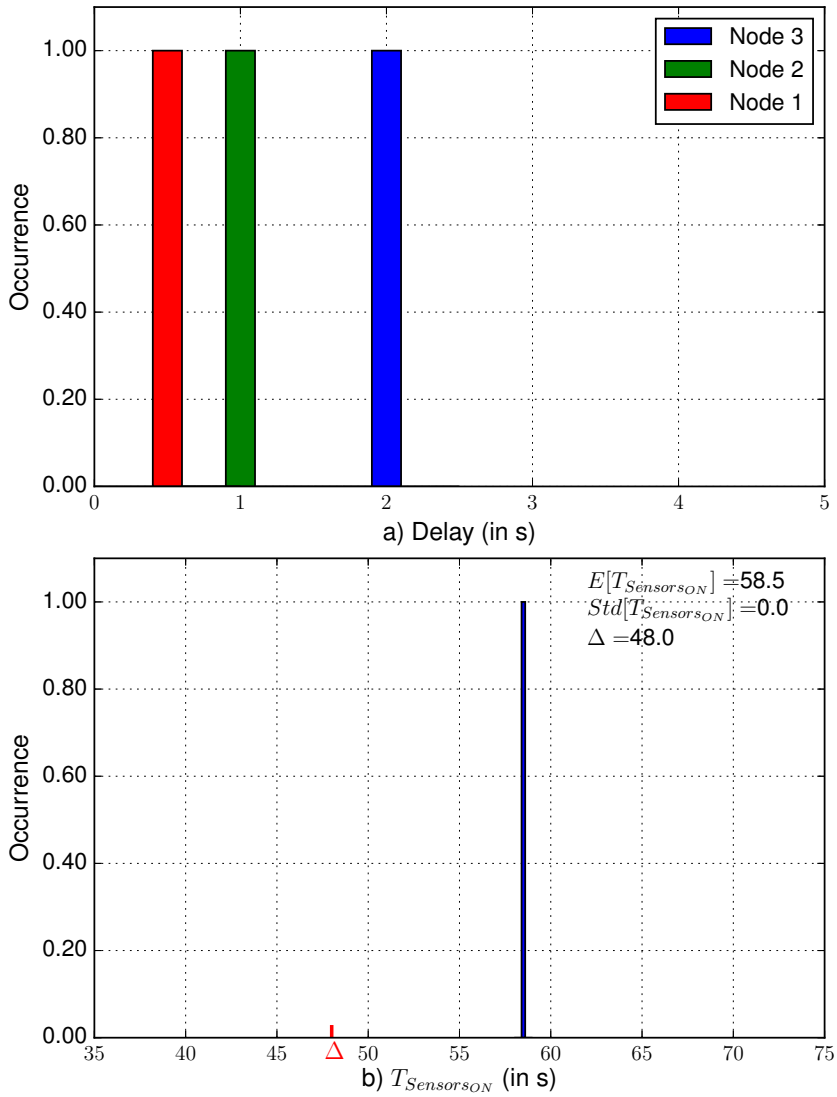

Figure 4.7: Constant network delays with $\alpha = 0.125$, $\beta = 10$. a) delay histogram; b) $T_{Sensors_{ON}}$'s histogram

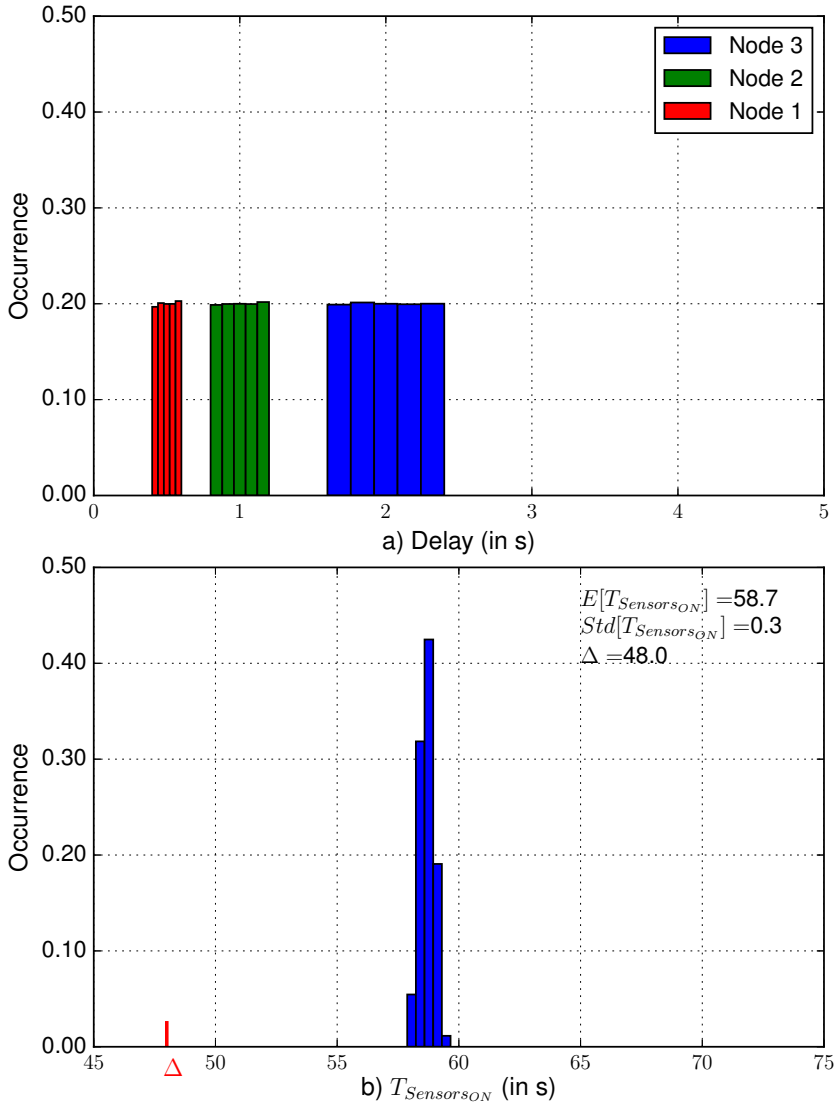

Figure 4.8: Uniformly distributed network delays. a) delay histogram; b) $T_{Sensors_{ON}}$'s histogram

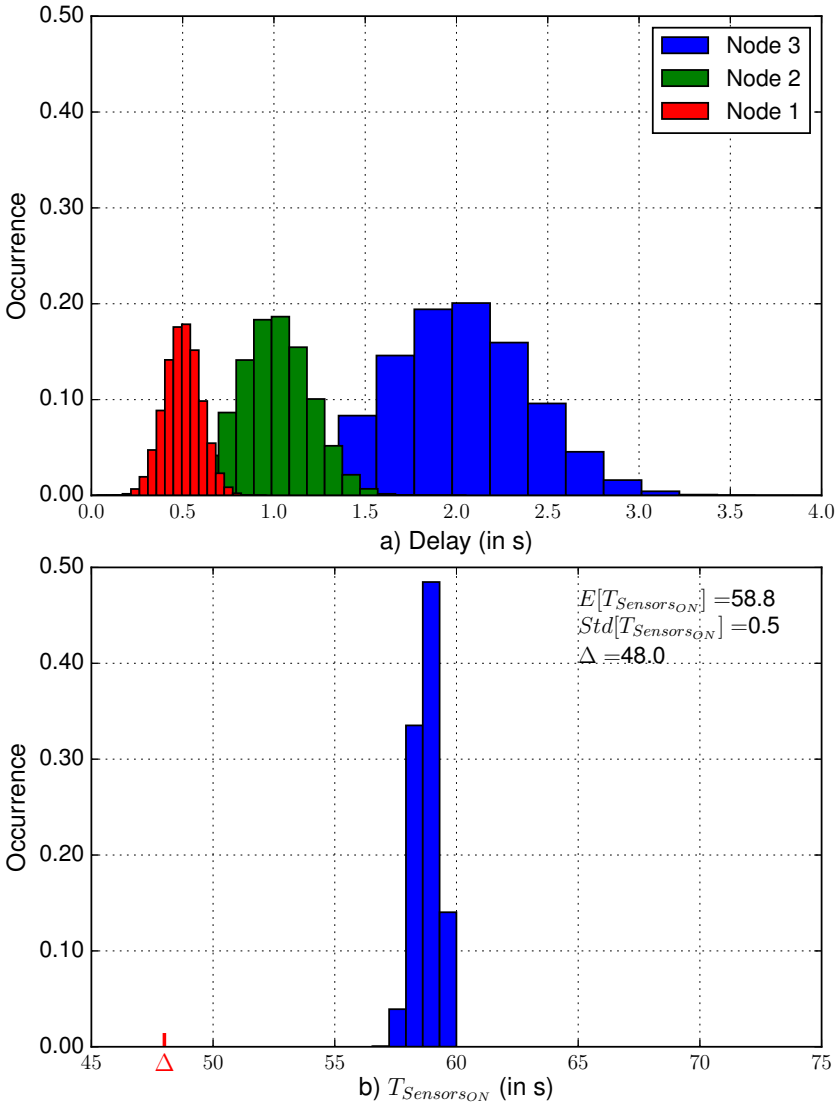

Figure 4.9: Gaussian distributed network delays. a) delay histogram; b) $T_{Sensors_{ON}}$'s histogram

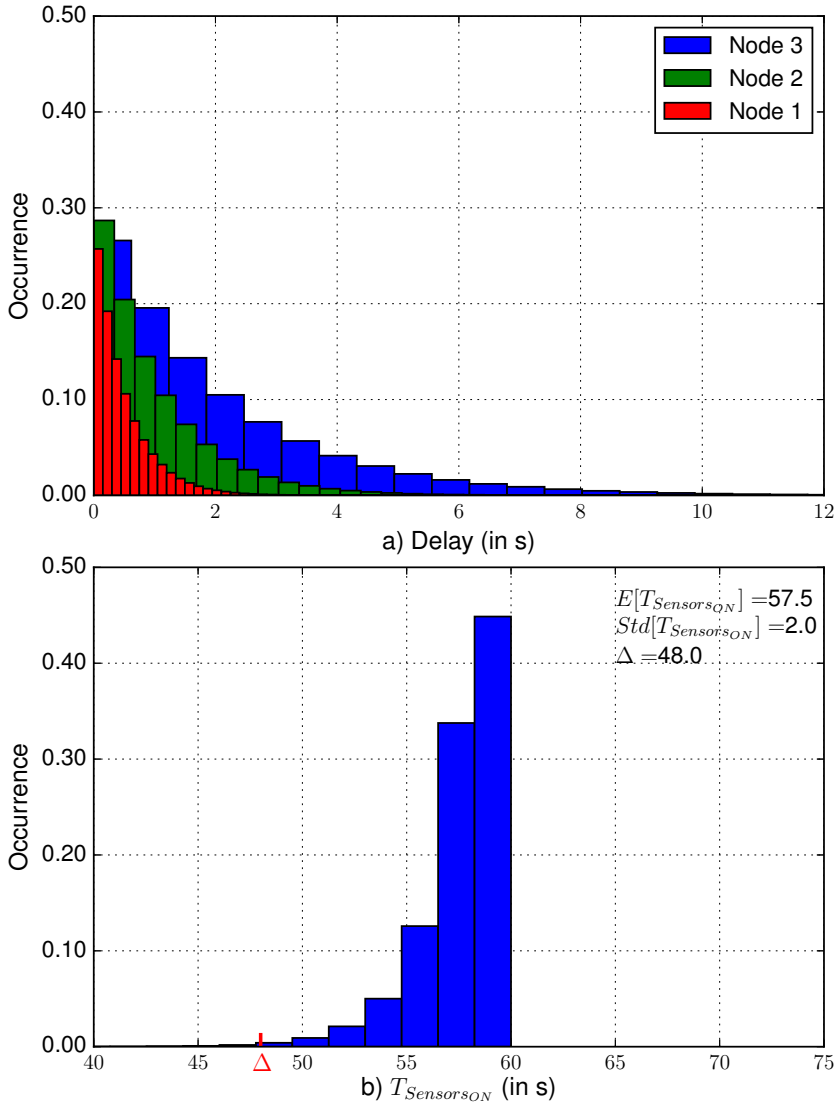

Figure 4.10: Exponentially distributed network delays. a) delay histogram; b) $T_{Sensors_{ON}}$'s histogram

case $E[T_{Sensors_{ON}}] = 57.52$ s and $T_{Sensors_{ON}} \in [25.06, 59.99]$ meaning that the nodes will maintain synchronism by about 99% of the cycles.

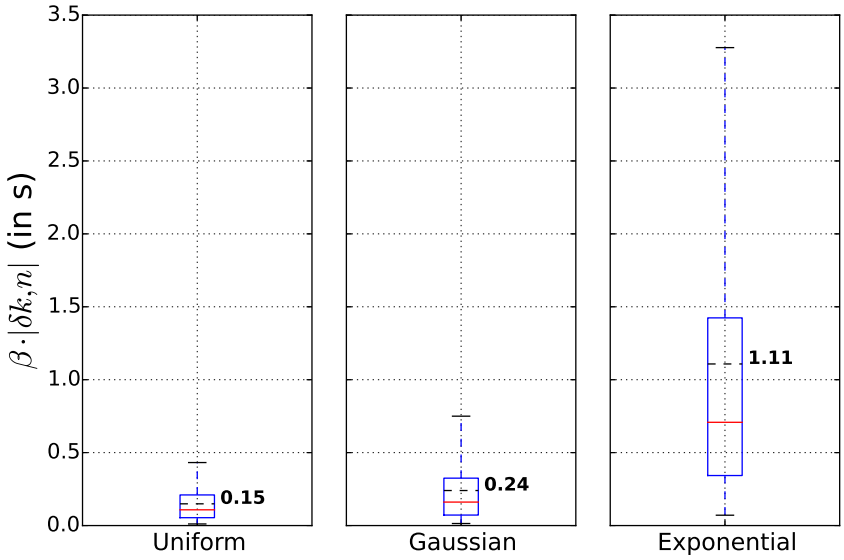

Figure 4.11: Box plot for $\beta \cdot |\delta_{k,n}|$, the sleeping offset represented in Eq. 4.2

Finally, Fig. 4.11 shows the box plot for the $\beta \cdot |\delta_{k,n}|$ component, which corresponds to the amount of time the nodes use to adjust sleep timers in order to wake up in synchronism in the next cycle. The worst value for the mean value of the $\beta \cdot |\delta_{k,n}|$ component is 1.11 s and it corresponds to the exponential distribution, what means that a node will not sleep during T_{OFF} s but, in average, will sleep during $T_{OFF} - 1.11$ s.

The box plot figures in this chapter give the standard metrics: the 25th percentile, the 75th percentile, and the red line is the median value. The top and bottom of the whiskers show the maximum and minimum values, respectively. Finally, the black dashed line in the box represents the mean value.

From this analysis we may conclude that the synchronization mechanism may be adequate for our purposes. In order to increase the trust in these results, a sensibility analysis is also carried out, in order to understand how $T_{Sensors_{ON}}$ is affected by different values of α and β.

4.2.1.1 α and β Values Estimation

We performed studies using different values for the synchronization mechanism parameters α and β. We considered 4 sensor nodes and assumed a uniformly distributed delays varying in $\pm 20\% \cdot 0.5$ s, $\pm 20\% \cdot 1.0$ s, $\pm 20\% \cdot 2.0$ s. Figures 4.12 to 4.20 show the results obtained when considering different values for the α and β parameters. Each of these figures present: a) the histogram of the network delays; b) the $T_{Sensors_{ON}}$'s histogram; c) the box plot for $\beta \cdot |\delta_{k,n}|$. The box plot gives the standard metrics: the 25th percentile, the 75th percentile, and the red line is the median value.

In the sensibility analysis shown below, we select two discrete set of values for α and β, $\alpha \in \{0.125, 0, 5, 0.875\}$ and $\beta \in \{10, 50, 100\}$. We vary one parameter at time while maintaining the other constant.

4.2.1.1.1 α Estimation: the weight given to the last sample in the calculation δ. Therefore, we want to investigate how it affects the synchronization mechanism by giving α different values, namely 0.125, 0.50, and 0.875.

4.2.1.1.2 β Estimation: since the $\delta_{k,n}$ value from Eq. 4.1 is small, we amplify it. The amplifying factor is the β parameter, and for it we selected three values, $\beta \in \{10, 50, \text{and } 100\}$.

Figures 4.12-4.20 show the results obtained for different combinations of the parameter's values. Table 4.1 summarizes it, showing: (a) the α and β values, (b) the $E[T_{Sensors_{ON}}]$, (c) average $E[T_{Sensors_{ON}}]$'s time in % of T_{ON}, and (d) $E[\beta \cdot |\delta_{k,n}|]$ component, the resulting sleeping offset. In this Table, *blue bold* values correspond to the ones that better satisfy our purposes.

Table 4.1: Summary of synchronization mechanism results as a function of α and β

Parameter		**Results**		
α	β	$E[T_{Sensors_{ON}}]$ **(in sec)**	$E[T_{Sensors_{ON}}]$ **(in % of** T_{ON}**)**	$E[\beta \cdot \lvert\delta_{k,n}\rvert]$
0.125	**10**	**58.7**	**97.82**	**0.149**
	50	59.0	98.33	0.745
	100	58.2	97.04	1.491
0.50	10	59.0	98.26	0.648
	50	55.6	92.59	3.241
	100	50.5	84.20	6.479
0.875	10	58.3	97.12	1.285
	50	50.3	83.87	6.428
	100	40.0	66.59	12.854

For the selection of the α and β values we considered the values that satisfy at same time: i) values of $T_{Sensors_{ON}}$ in % of T_{ON} above 80%; and ii) lowest $\beta \cdot |\delta_{k,n}|$ component value.

4.2.1.2 Results and Discussion

This analysis of the results showed that not all the values chosen for α and β parameters satisfy our synchronization mechanism requirements. In fact, if we consider respectively $\alpha = 0.50$ and $\beta \in \{50; 100\}$ the mechanism will fail because the probability $P[T_{Sensors_{ON}} \geq \Delta] < 1$ (see respectively Fig. 4.16 and 4.17, what means that the sensor nodes will not be synchronized in all their duty cycles. The same applies if we consider $\alpha = 0.875$ and $\beta \in \{50; 100\}$, as shown in Figures 4.19 and 4.20. From c) we can observe that there are occurrences for $T_{Sensors_{ON}}$ below 80%, the threshold established for success, being in average equal to 97.82% of T_{ON}. Therefore, those values do not satisfy our selection criteria. From the other values evaluated we may consider that

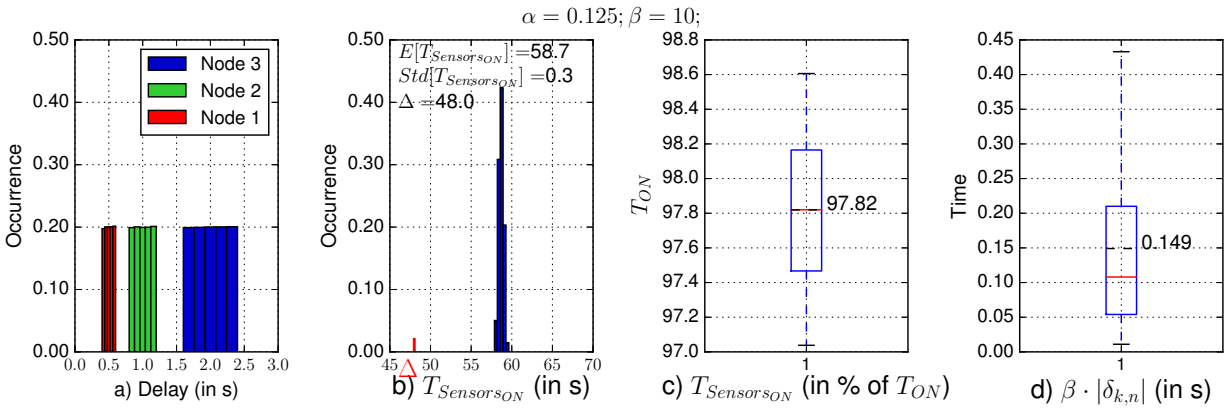

Figure 4.12: Uniformly distributed network delays with α=0.125; β=10. a) delay histogram; b) $T_{Sensors_{ON}}$'s histogram; c) Box plot for $T_{Sensors_{ON}}$ (% of T_{ON}); d) Box plot for $\beta \cdot |\delta_{k,n}|$

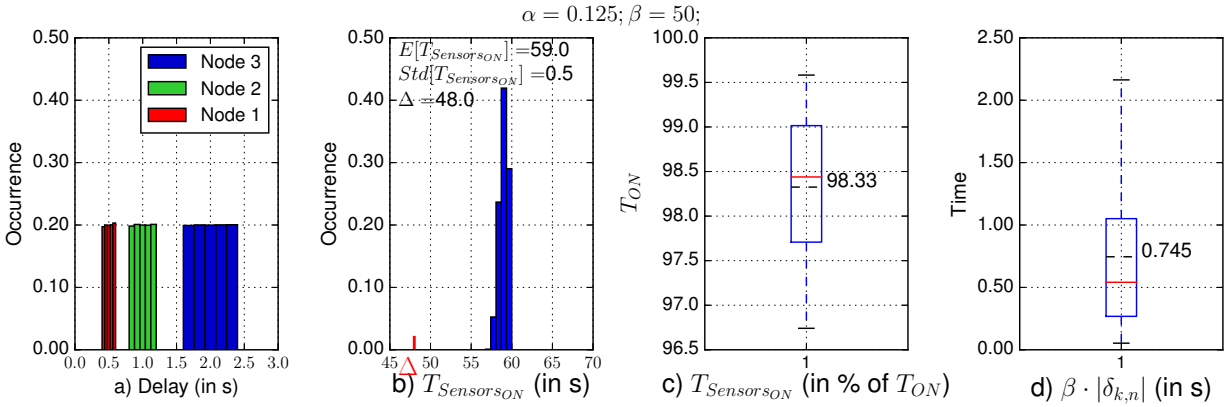

Figure 4.13: Uniformly distributed network delays with α=0.125; β=50. a) delay histogram; b) $T_{Sensors_{ON}}$'s histogram; c) Box plot for $T_{Sensors_{ON}}$ (% of T_{ON}); d) Box plot for $\beta \cdot |\delta_{k,n}|$

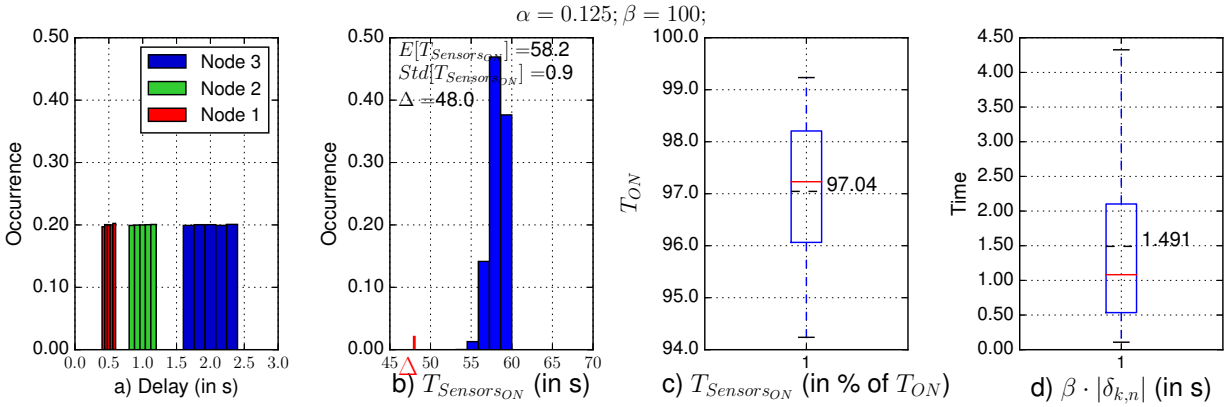

Figure 4.14: Uniformly distributed network delays with α=0.125; β=100. a) delay histogram; b) $T_{Sensors_{ON}}$'s histogram; c) Box plot for $T_{Sensors_{ON}}$ (% of T_{ON}); d) Box plot for $\beta \cdot |\delta_{k,n}|$

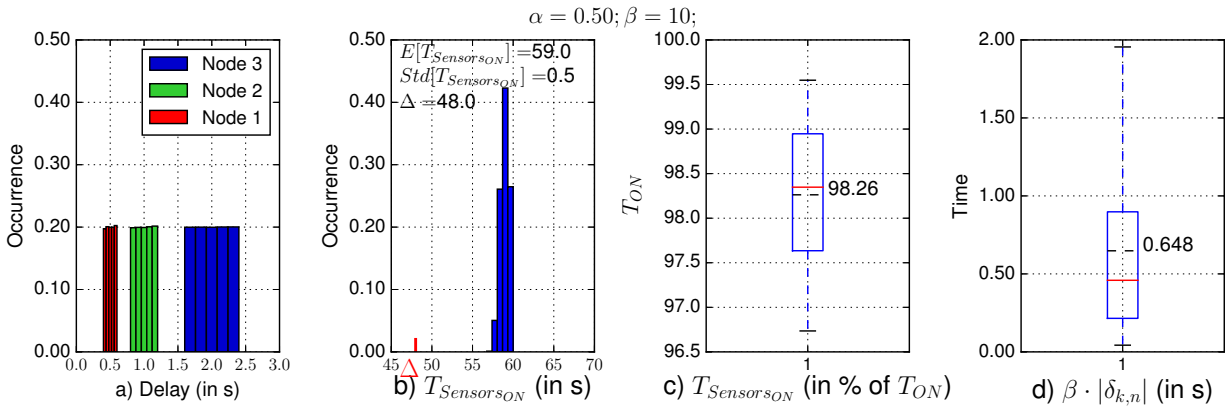

Figure 4.15: Uniformly distributed network delays with α=0.50; β=10. a) delay histogram; b) $T_{Sensors_{ON}}$'s histogram; c) Box plot for $T_{Sensors_{ON}}$ (% of T_{ON}); d) Box plot for $\beta \cdot |\delta_{k,n}|$

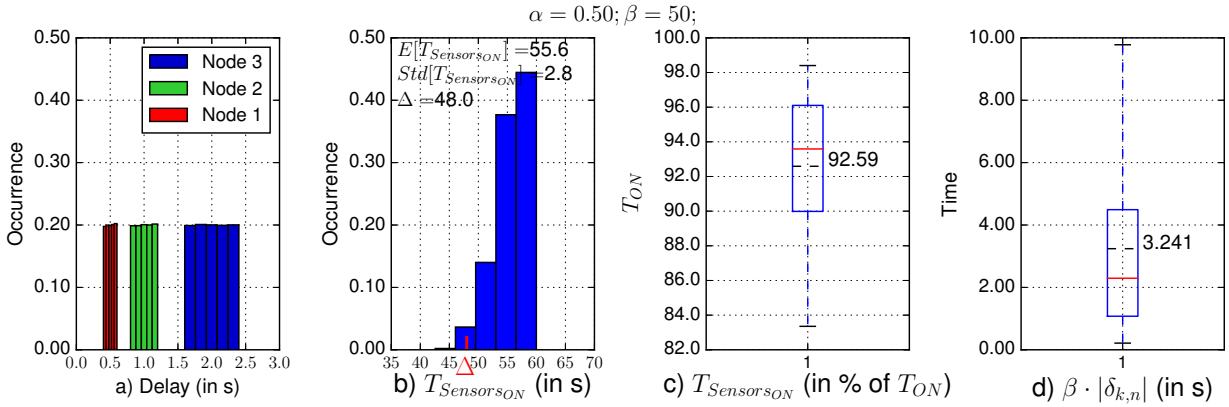

Figure 4.16: Uniformly distributed network delays with α=0.50; β=50. a) delay histogram; b) $T_{Sensors_{ON}}$'s histogram; c) Box plot for $T_{Sensors_{ON}}$ (% of T_{ON}); d) Box plot for $\beta \cdot |\delta_{k,n}|$

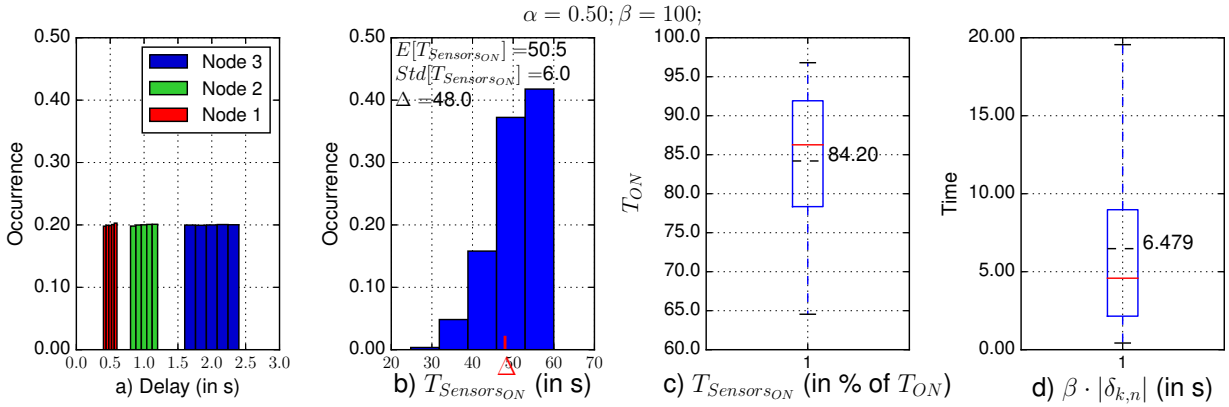

Figure 4.17: Uniformly distributed network delays with α=0.50; β=100. a) delay histogram; b) $T_{Sensors_{ON}}$'s histogram; c) Box plot for $T_{Sensors_{ON}}$ (% of T_{ON}); d) Box plot for $\beta \cdot |\delta_{k,n}|$

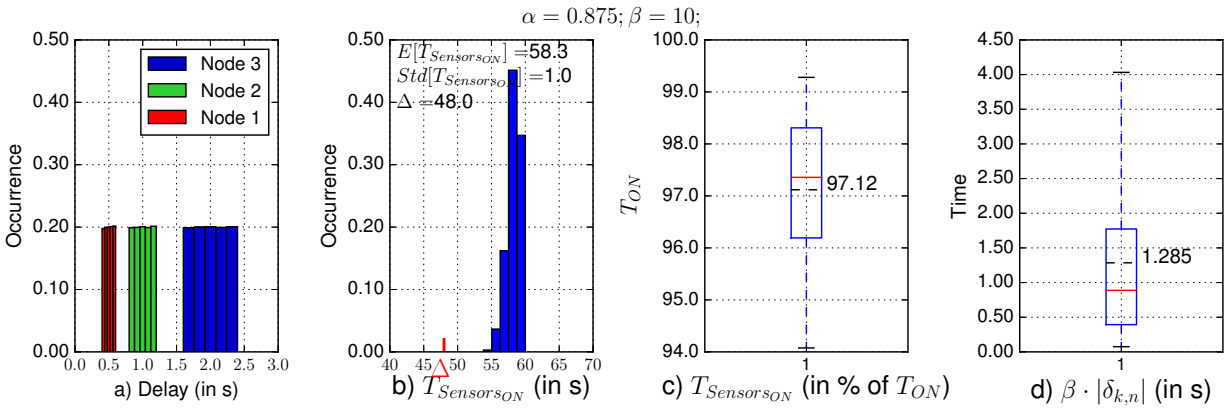

Figure 4.18: Uniformly distributed network delays with α=0.875; β=10. a) delay histogram; b) $T_{Sensors_{ON}}$'s histogram; c) Box plot for $T_{Sensors_{ON}}$ (% of T_{ON}); d) Box plot for $\beta \cdot |\delta_{k,n}|$

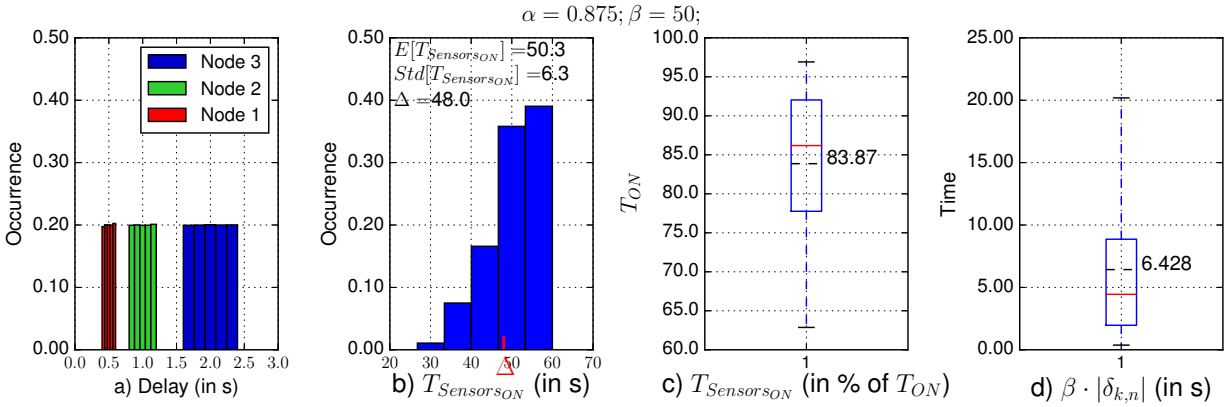

Figure 4.19: Uniformly distributed network delays with α=0.875; β=50. a) delay histogram; b) $T_{Sensors_{ON}}$'s histogram; c) Box plot for $T_{Sensors_{ON}}$ (% of T_{ON}); d) Box plot for $\beta \cdot |\delta_{k,n}|$

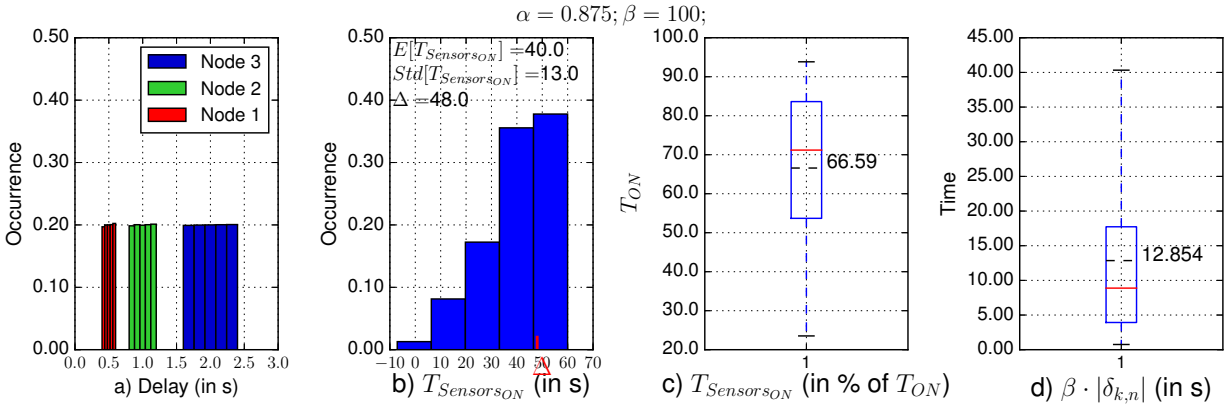

Figure 4.20: Uniformly distributed network delays with α=0.875; β=100. a) delay histogram; b) $T_{Sensors_{ON}}$'s histogram; c) Box plot for $T_{Sensors_{ON}}$ (% of T_{ON}); d) Box plot for $\beta \cdot |\delta_{k,n}|$

$\alpha = 0.125$ and $\beta = 10$, are the values that better satisfy our purposes, for the scenarios considered. Comparing to other pair os values for α and β, these values present at same time: i) greater $T_{Sensors_{ON}}$ value (58.7), which is almost the same theoretical value of T_{ON}; ii) all the occurrences for $T_{Sensors_{ON}}$ in terms of $T_{ON}\%$ are above 97%, being in average equal to 97.82%; and iii) the mean $\beta \cdot |\delta_{k,n}|$ component has, in average, the lowest value 0.149 what means that the sensor nodes have to wakeup before the next duty cycle less time than in the other cases. This will have impact in energy consumption since the sensor nodes do not have to stay unnecessary time awaked.

4.2.2 Simulations

In [14] two different applications were used in three different scenarios, being the nodes distributed as shown in Fig. 4.21. Simulations ran in Contiki Cooja simulator [120]. All the nodes are within a distance of 25 meters for a transmission range of 30 meters, and support one of the two applications. Each application is running in eight nodes, and each node runs a single application. In *Scenario 1* the nodes running *App. A* were selected in a way that a long path could be obtained; in *Scenario 2* both applications have the same node distribution; *Scenario* 3 is used to investigate situations where at least one node from other application is required to relay data. In the scenarios simulated, sink nodes are always awake, and sink node running App. B (node 9) was chosen as the network DAG root because of its application duty-cycle. For the nodes running *Application A*, $T_{ON} = 60$ s, $T_{OFF} = 3540$ s; for the nodes running *Application B*, $T_{ON} = 60$ s, $T_{OFF} = 840$ s.

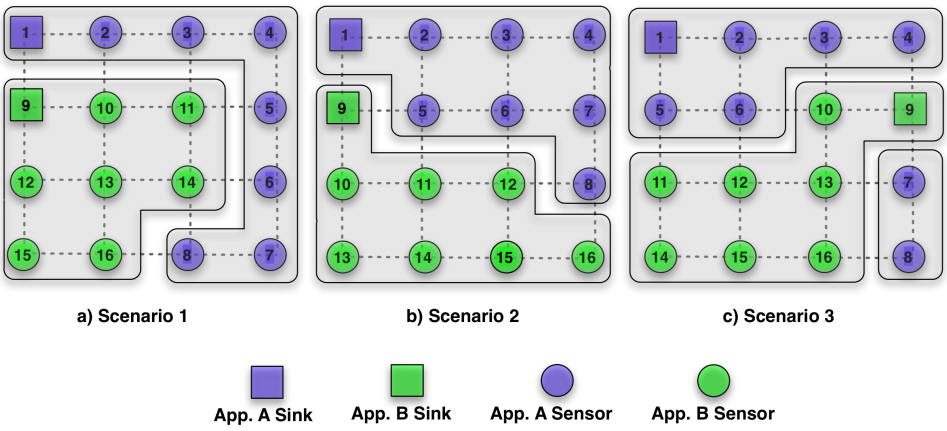

Figure 4.21: Scenarios simulated

We simulated two situations: i) a situation where all the nodes join the network at same time, so that the proposed synchronization mechanism is not used as, in simulations with *COOJA*, clock drifting is the same for all the sensor nodes; and ii) the nodes will join the network at different time. The later implies the use of the synchronization mechanism described in Sec. 4.1 in order to keep the nodes synchronized with respect to the applications they run. The nodes join the network at different times which were randomly generated between 317 and 1102 s.

4.2.2.1 Results and Discussion

Each time a node receives a query it computes the time it must wakeup before the start of the next application cycle in order to be able to receive and forward packets, and to reply back to the sink successfully. The synchronization mechanism was configured with $\alpha = 0.125$ and $\beta = 10$.

In the the simulations 16 nodes have been used, half of them running each application. Each scenario was simulated ten times and information was extracted in order to estimate delays, *QSR*, energy, and the $E[\beta \cdot |\delta_{k,n}|]$ component. The results obtained are the following.

4.2.2.1.1 Delays: we considered *delay* as the sum of all per-hop delays for each sensor query packet reception and characterized by the sum of the processing and queueing delays in intermediate and destination sensor nodes, and the transmission delays and propagation delays in intermediate nodes. Fig. 4.22 shows the nature of the delays observed and, as it can be seen, network delays are uniformly distributed.

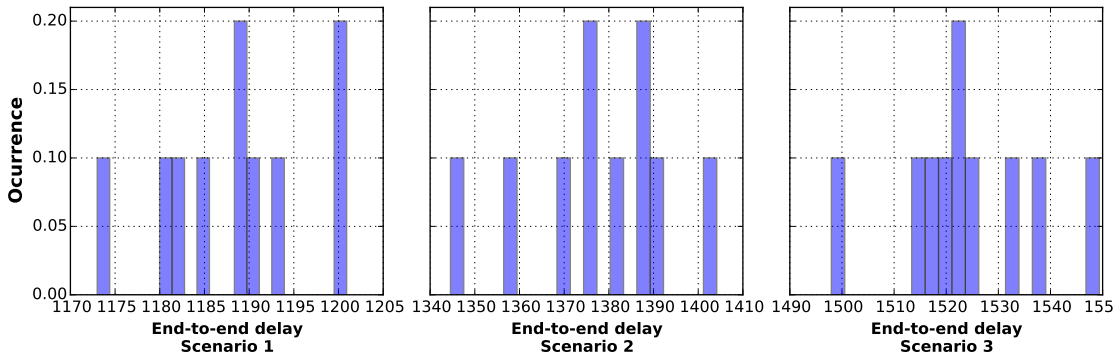

Figure 4.22: Mean delays histogram, for each scenario (in ms)

4.2.2.1.2 Per Hop $\beta \cdot |\delta_{k,n}|$ Component: from the simulations we have extracted information about the $\beta \cdot |\delta_{k,n}|$ component on a per hop basis. Fig. 4.23 shows the box plot for this component in scenario 1. We can observe that, except for the first hop, this component presents per hop similar values, and sensor nodes would have to wake with an average sleeping offset of about 0.232 s. In the first hop the sleeping offset has a grater value (0.49 s in average) because in this hop we can observe some congestion, particularly between the sink node (node 1) and the sensor node 2.

Fig. 4.24 shows the box plot for the $\beta \cdot |\delta_{k,n}|$ component in scenario 2. As in scenario 1, we can also se that this component presents similar values per hop, with an average sleeping offset of about 0.176 s.

Finally, Fig. 4.25 shows the box plot for the $\beta \cdot |\delta_{k,n}|$ component in scenario 3. As in the other two scenarios, we observed that this component presents similar values per hop, in a average about 0.242 s. In the case of the sleeping offset for 2 hop nodes, it has in average a grater value (0.35 s). Analyzing this scenario's topology, and the traffic that may occur, we can observe some congestion around sensor nodes 3 and 13. For sensor node 3 it needs to forward replies from

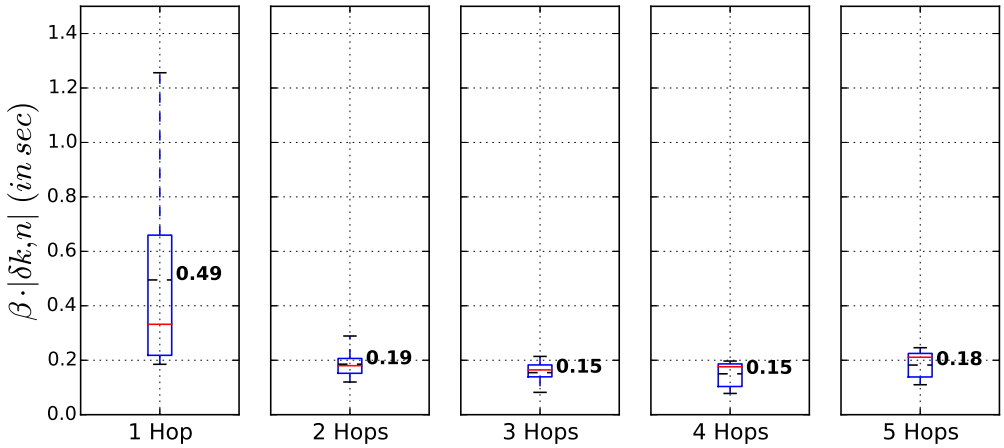

Figure 4.23: Box plot for $\beta \cdot |\delta_{k,n}|$, with α=0.125 and β=10, for each hop in scenario 1

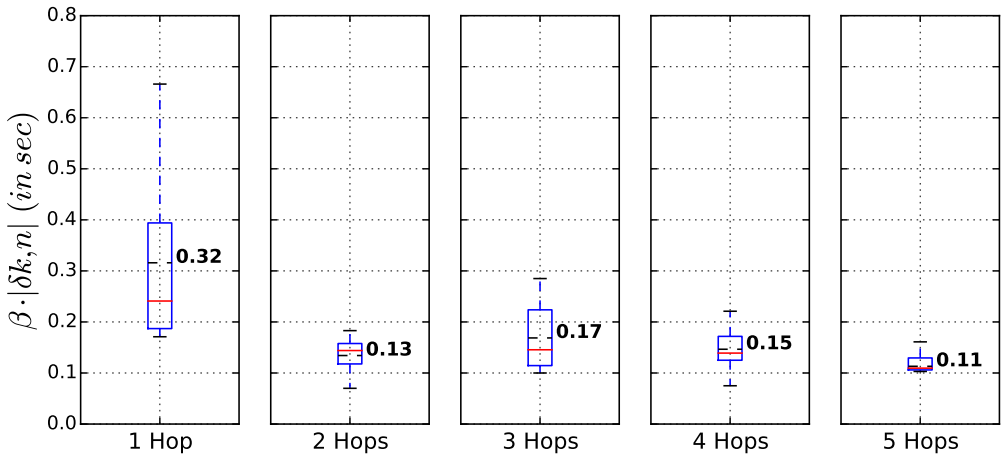

Figure 4.24: Box plot for $\beta \cdot |\delta_{k,n}|$, with α=0.125 and β=10, for each hop in scenario 2

sensor nodes 4, 7 and 8. For sensor node 13 it also forwards replies from sensor nodes 11, 12, 14, 15 and 16. However, this sleeping offset value can be also considered as negligible as has a small additional value.

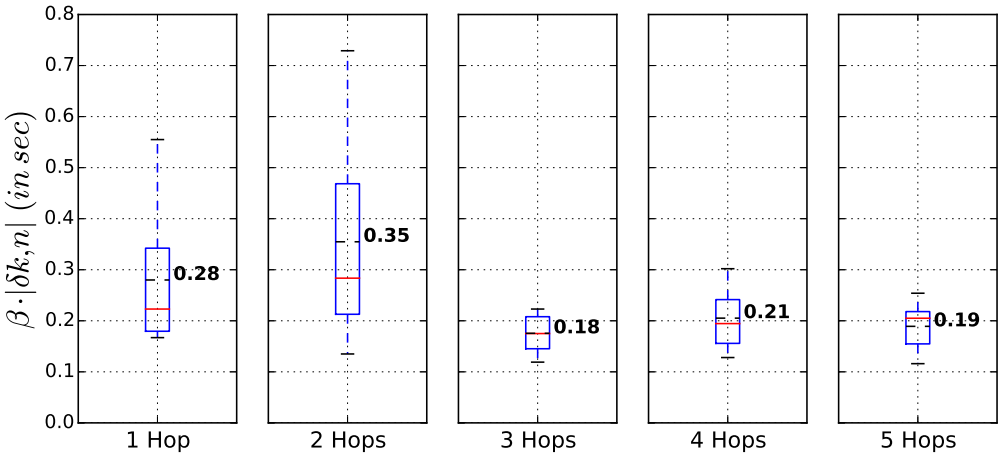

Figure 4.25: Box plot for $\beta \cdot |\delta_{k,n}|$, with α=0.125 and β=10, for each hop in scenario 3

Fig 4.26 shows the histogram of the $\beta \cdot |\delta_{k,n}|$ component. As it can be seen, this sleeping offset component accepts values between 0.40 s and 0.75 s. In this interval, the component has a uniform behavior.

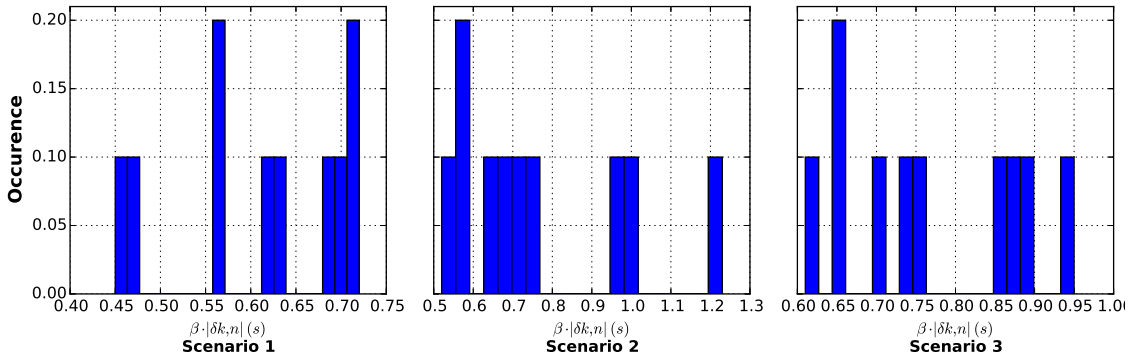

Figure 4.26: $\beta \cdot |\delta_{k,n}|$, with α=0.125 and β=10, histogram for each scenario (in sec)

In Fig 4.27 is shown the box plot for the expected sleeping offset value for each of the three scenarios studied. As it can be verified, the nodes would sleep not the T_{OFF} time, but in average $T_{OFF} - 0.716$ s. Moreover, we observed that, independently of the network topology, this component has almost the same values, what confirms that the synchronization mechanism proposed is adequate for our purposes. Moreover, comparing the results from Fig 4.26 and Fig 4.27 we observe that for scenarios 2 and 3 the maximum and minimum $\beta \cdot |\delta_{k,n}|$ values are different. In scenario 2 the nodes have more neighbors running the same application, what implies that each of them may need more time to access the wireless medium to forward a query. This is also reflected on the network delays and affects the $\beta \cdot |\delta_{k,n}|$ component.

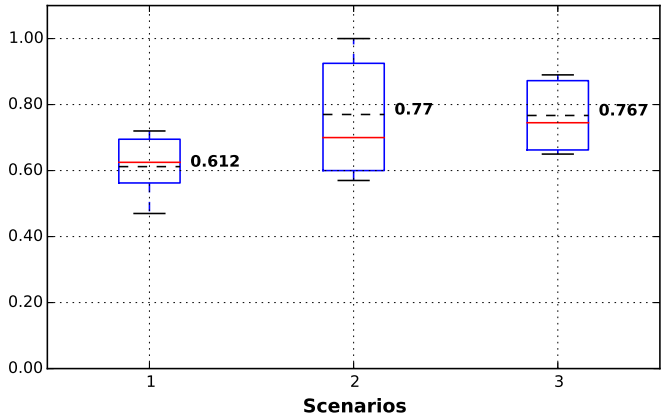

Figure 4.27: Box plot for $\beta \cdot |\delta_{k,n}|$, with α=0.125 and β=10, for each scenario (in sec)

4.2.2.1.3 QSR: Fig. 4.28 shows the box plot for *QSR*. In this figure is showed: (1) the results using the standard *RPL* routing protocol, (2) the results using *RPL-BMARQ* solution proposed in [14] without the synchronization mechanism implemented, and (3) the results using the same *RPL-BMARQ* solution fully implemented. As it can be observed, in average, 98.8% of the queries sent by sinks are replied by sensor nodes. With this success ratio, we can argue that the quality of the proposed synchronization mechanism is confirmed.

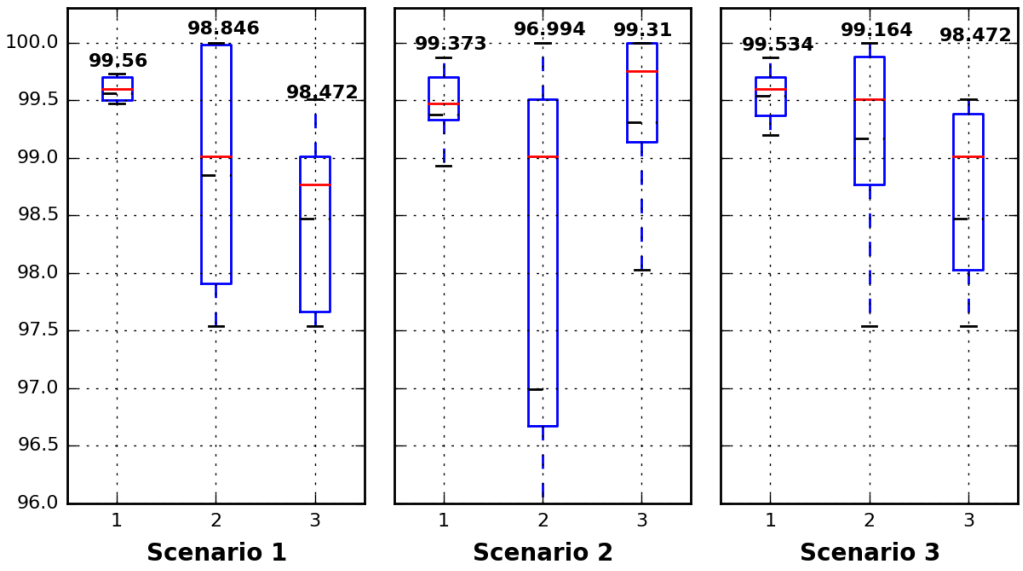

Figure 4.28: Mean Query Success Ratio - QSR for each scenario (in %)

4.2.2.1.4 Energy: in order to estimate the impact of the synchronization mechanism in the nodes energy consumption, we considered energy consumption related only to communication aspects: packet transmission, reception, *radio "idle"* the state where a node has its *radio on* and is waiting to send or to receive a data "packet", and *radio interferences*, as shown in Eq. 3.6.

Fig. 4.29 shows the energy consumed by the nodes in three situations evaluated for each scenario: i) using the standard *RPL*; 2) using *RPL-BMARQ* without the implementation of the synchronization mechanism; and 3) using *RPL-BMARQ* with the synchronization mechanism implemented. In this figure we can observe that the nodes using *RPL* solution consume much more energy than using *RPL-BMARQ*. The *RPL* solution does not perform radio duty cycling, and does not put the sensor nodes to sleep when there is no activity associated to them. *RPL* must rely on lower layers to perform radio duty cycling. On the other hand, *RPL-BMARQ* performs radio duty cycling, putting asleep, and wakening the nodes according the application duty cycles they run. The figure also shows that with the *RPL-BMARQ* solution fully implemented we have much lower power consumption, compared to energy spent by the nodes when the *"standard" RPL* is used. On the other hand, if we use *RPL-BMARQ* without the synchronization mechanism, the energy consumption is even lower because the nodes wakeup exactly when the next application duty cycle starts, and sleep also exactly at same time.

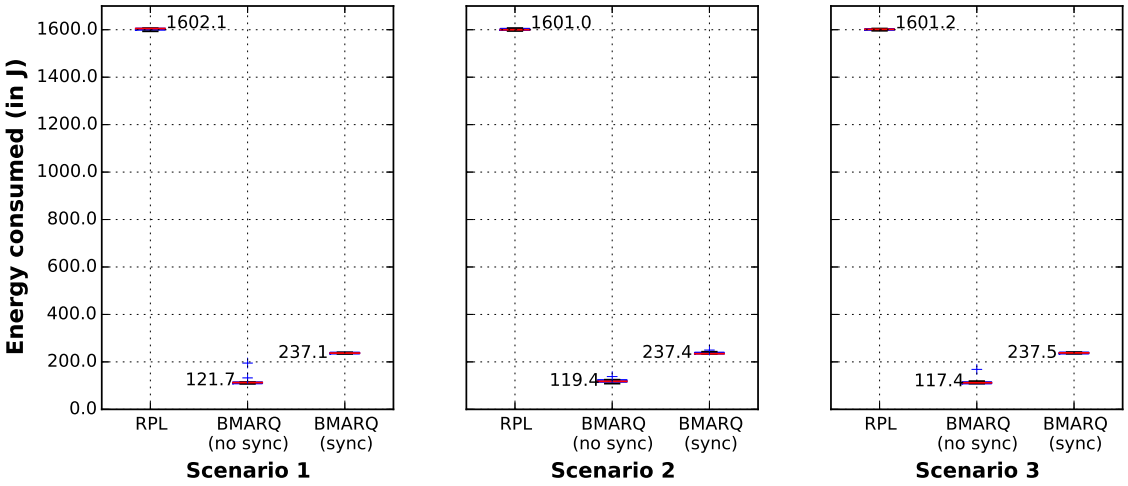

Figure 4.29: Energy consumed by each solution in each scenario

4.2.3 Testbed Experiments

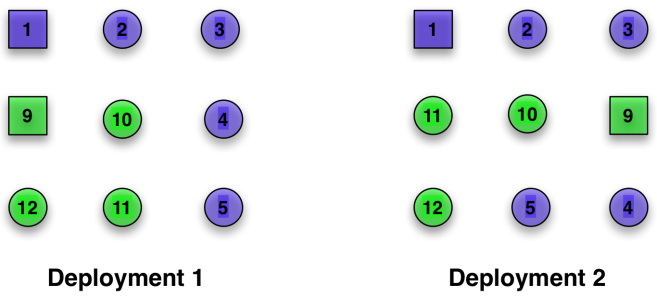

Figure 4.30: Scenarios deployed

In order to confirm the results obtained from theoretical studies and simulations, we also tested our proposed solution in a real environment. For that purpose, two of the scenarios studied were selected (scenarios 1 and 3) and deployed. Since it was not possible to reproduce them at same scale, the scenarios deployed correspond to a 3x3 square lattice topology, while keeping all the same assumptions. In order to obtain reliable terms of comparison, we have simulated these deployments using the same methods as in section 4.2.2 and compared the simulated results with those obtained in testbeds.Fig. 4.30 shows both deployments, which were realized using *TelosB motes* [18] (see Fig. 2.11), placed at distances of 5 meters, and the radio transmission power was reduced to -7 dBm in order to reduce nodes radio influence space. Application A runs in five nodes (1, 2, 3, 4 and 5), while application B runs in four nodes (9, 10, 11 and 12). Node 9 is, at the same time, the root of the DAGs and a sink. Node 1 is the other sink. The nodes were coded using *ContikiOS* (2.6)[16]. *ContikiOS* is a Operating System for WSN which incorporates a implementation of the *IPv6* protocol stack and uses *RPL* as the default routing protocol.

4.2.3.1 Results and Discussion

Each experiment was carried out for 4 hours. To log real time data, two *Raspberry Pi* platforms were used, connected to both sink nodes via a serial connection. Inside each Raspberry Pi [121] platform was a python program running, responsible to get timestamp data from each sink with respect to query packets sent and reply packets received. In order to verify our proposed synchronization mechanism we considered in this work: i) synchronization parameter's values α=0.125 and β=10; ii) packet reception time on the sink nodes side to estimate the expected reception time and to compute the sleeping offset component ($\beta \cdot |\delta_{k,n}|$); iii) *QSR* results. The main results obtained include the following:

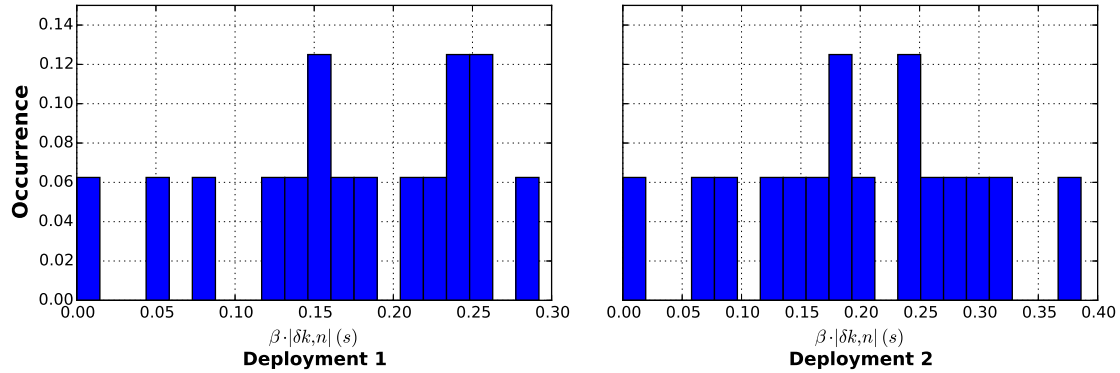

Figure 4.31: $\beta \cdot |\delta_{k,n}|$, with α=0.125 and β=10, histogram for each deployment (in %)

4.2.3.1.1 $\beta \cdot |\delta_{k,n}|$ Component: Fig. 4.31 shows $\beta \cdot |\delta_{k,n}|$ component, the sleeping offset represented in Eq. 4.2) histogram for each deployment. As expected it presents the same uniform distribution characteristics as the theoretical evaluation, and the simulations performed. Moreover, we can see in Fig. 4.32 that this component presents in average a sleeping offset of 0.185 s.

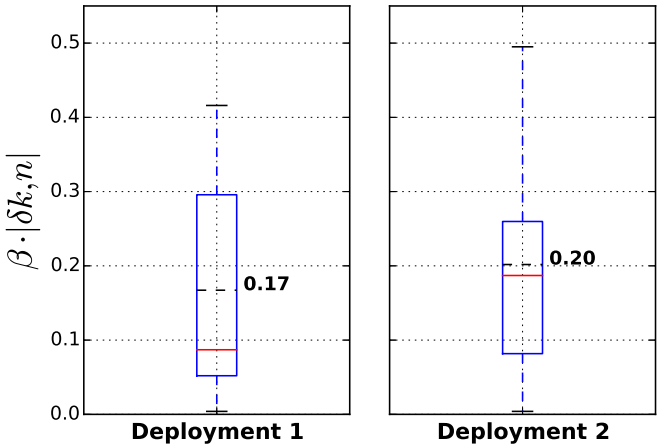

Figure 4.32: Box plot for $\beta \cdot |\delta_{k,n}|$, with α=0.125 and β=10, for each deployment (in sec)

4.2.3.1.2 QSR: Fig. 4.33 shows simulation and real implementation results. As it can be seen, both present same values (100%), which means that also in real testbeds, the nodes reply to all the queries sent by sinks, going to sleep and waking up while being synchronized.

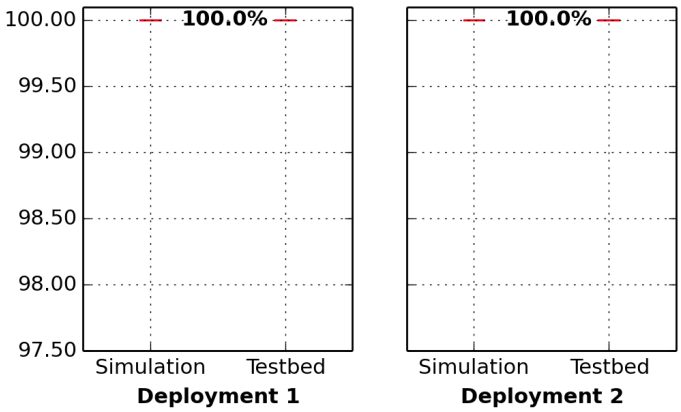

Figure 4.33: Query Success Ratio (QSR) for the scenarios deployed (in %)

From the above results we can conclude that there are no major differences between what was observed in the theoretical studies, in the simulation environment, and what was expected in the testbed environment. This confirms the usability and the quality of the synchronization mechanism proposed, when applied to *Application-driven WSNs* with the characteristics described in this work.

In contrast to our centralized and synchronous proposed synchronization mechanism, in [122] is proposed a synchronous protocol that provides a distributed strategy which guarantees convergence for any undirected connected communication graph. This strategy tries to control the nominal clock period and the clock offset based on the information received from neighbor nodes in order to achieve synchronization. Moreover, when a underlying communications graph is known, the authors purpose an optimal design strategy which can be used to study the effect of noise and external disturbances on the steady-state performance.

The integration of WSN with the IoT raises security issues as these networks may be subject of malicious attacks. In contrast to our work, which does not address security in WSN, in [123] is proposed the *average-consensus-based time synchronization protocol* (ATS) which does not depend on any reference node or network topology, making it robust to different kinds of network attacks. ATS is vulnerable to message manipulation attacks but the authors propose solutions to overcome them. Firstly, they investigate the impact of message manipulation attacks and derive a necessary condition for ATS to converge. Secondly, authors propose a new adjusting parameter which exploits the two-hop neighboring information to constrain the attackers. Finally, the authors incorporate all checking processes into a *secure average-consensus-based time synchronization protocol* (SATS). This work proves that SATS guarantees the network time synchronization with an exponentially converging speed.

In [124] WSN time synchronization follows two strategies: i) *Maximum Time Synchronization* (MTS) to simultaneously synchronize the skew and offset of each node when the communication delay is negligible, and ii) a *Weighted Maximum Time Synchronization* (WMTS) when the communication delay between the nodes is random. In contrast to our work, in which we synchronize a virtual clock, these authors attempt to synchronize the clock skew, in order to obtain acceptable synchronization accuracies. The main idea of MTS and WMTS is to drive all clocks to the maximum value among the network. In [124] are considered random communication delays with normal distribution, while we validated our solutions against gaussian and exponentially distributed delays. This solution can be classified as distributed and asynchronous algorithm, whereas ours can be classified as centralized and synchronous.

4.3 Summary

This chapter describes our second contribution. In Section 4.1 we present and describe the proposed *Application-Driven WSN Synchronization Mechanism* to maintain synchronized all the sensor nodes in WSNs. The mechanism uses the *EWMA* technique to control the time the sensor nodes would wake before the start of a new application cycle. In Section 4.2 an evaluation of the synchronization mechanism is made, in which it was assumed that the sensor nodes are affected by different network delay distributions. In Section 4.2.2 we performed simulations using *ContikiOS* and *Cooja* to evaluate the mechanism in different network topologies, and confirmed its functionalities.

Finally, two real testbeds were implemented and the experiments confirmed our simulations results, showing that the mechanism also works in real applications.

Chapter 5

Conclusion

In this thesis we developed the *RPL-BMARQ* communications solution that can be used to interconnect a network of sensor nodes. The solution extends the *RPL* routing protocol with the purpose of improving energy efficiency by making the network aware of the traffic generated by its applications. The developed solution assumes that sensors form an *IPv6* network, each sensor is enabled to run a single application, and the sensor nodes are not always active. By considering the neighbors of each node and the application each node runs, the developed communications solution enables the data of every application and node to be transferred while keeping the overall energy consumed by the network low. The sensor nodes are expected to join the network at different times and they are not synchronized. Therefore, a *synchronism mechanism* for the developed communications solution is also proposed.

5.1 Work Review

Chapter 2 describes the *IEEE 802.15.4 standard*, 6LoWPAN, routing protocols for IPv6 based WSNs, operating systems and simulation environments for WSN, and sensor hardware platforms. This chapter also summarizes the state of the art on Energy efficiency and node synchronization in Wireless Sensor Networks. Each sensor device in a WSN has hardware and software constraints and is equipped with its own local clock, presenting synchronization challenges because of the distributed nature of these networks. Therefore, the chapter also presents, describes and discusses the state of the art regarding *time synchronization in WSN*. Finally, this chapter presents and discusses techniques related to wakeup mechanisms that try to increase the nodes lifetime, namely *duty cycling techniques* and *scheduled rendezvous techniques*.

Chapter 3 describes the *RPL-BMARQ* communications solution, an extension to the *RPL* routing protocol that provides the creation of the *DAG*s following the *Application-Driven WSN* concept. The solution tries to insure that data of an application is relayed mainly by the sensor nodes running that application. The DAGs are created and maintained by choosing mainly the nodes running the same application as parent; the nodes not associated to that application will not be selected as parent. The solution was evaluated against *standard RPL*. For this purpose

four network scenarios with different network topologies were selected and several simulations were performed. Obtained results were analyzed considering: (1) DAGs created, (2) energy consumption, (3) *Query Success Ratio*, (4) QSR fairness, (5) delay, (6) number of packets per query, and (7) reaction to topology changes. The results showed that *RPL-BMARQ*: (1) follows the *Application-Driven* concept and constructs the DAGs differently from the *RPL* solution; (2) presents major energy gains with, in average, values of about 92%; (3) presents a *QSR* of 98%, similar to the *RPL* solution; (4) presents, in average, QSR fairness index values for both solutions above 99%; (5) makes delays to increase about 10%, in average, but our claims on energy saving still holding; (6) presents lower total number of packets sent and received than the equivalent RPL; and (7) presents a behavior just like *RPL* with respect to convergence times. In order to confirm the results obtained from simulations, we also tested *RPL-BMARQ* in a real environment. For that purpose, two of the scenarios studied were selected and deployed. The real power consumption was measured and compared to the energy consumption obtained from simulations.

Chapter 4 proposes an *Application-Driven WSN node synchronization mechanism* to synchronize the sensor nodes according to the application cycle they run, since it is unlikely that all the sensor nodes would join a WSN at the same time. Therefore, all the sensor nodes need to be awaken almost at same times in each application cycle in order to receive sink queries and forward them to the other nodes. This synchronization mechanism uses a synchronous method which includes a *synchronization setup phase* and a *synchronization maintenance phase*. The mechanism makes use of the *Exponentially Weighted Moving Average* (EWMA) technique in order to control the time a sensor node would wakeup in the next cycle. The work presents a study of this mechanism assuming that the sensor nodes are affected by different network delay distributions, allowing the nodes to go asleep and to wakeup in synchronism. A theoretical evaluation of the mechanism is performed in order to estimate the α and β synchronization parameters, responsible to control the time a sensor node need to wakeup in the next cycle before the reception of a packet. To this end, different types of network delays (constant, uniform, gaussian and exponential) were studied in order to observe the behavior of the mechanism and help selecting the better values for α and β. Finally, the mechanism was evaluated by means of simulations, and confirmed its functionalities.

5.2 Contributions Summary

This thesis provides two major original contributions:

- **A novel mechanism using application-layer topologies (*RPL-BMARQ*) to constrain RPL-defined routing trees**: a communications stack named *RPL-BMARQ* is proposed to insure that data of an application is relayed mainly by the sensor nodes running that application. At this end, the solution changes *RPL* in order to change how the network DAGs are created and maintained, by choosing mainly the nodes running the same application as parent. *RPL-BMARQ* assumes that every node will primarily select its parent from a set of nodes running the same application to which the data is associated, mainly exchanging

packets between neighbors running the same application, avoiding paths which may include nodes that do not run this application. Also, the solution puts the sensor nodes asleep when there is no activity related to their applications. Parts of this contribution have also been presented in [13, 14, 15].

- **A novel Application-Driven WSN node synchronization mechanism for RPL-BMARQ**: the sensor nodes must share some kind of time reference which allow them to be synchronized with respect to the life cycle of the applications they run. Therefore, an *application-driven synchronization* mechanism is proposed which will help the nodes running the same application to wakeup and to go asleep in a synchronized manner. The mechanism makes use of the EWMA technique for calculating run-time average network delays to control the time the sensor nodes would sleep before the next cycle, so they can successfully send, receive, and forward packets. Parts of this contribution have also been presented in [15].

5.3 Future Work

Future work related to this work may include the following topics:

- **Support for multiple applications**: In the scenarios evaluated the network and the nodes support two different applications, running one application per sensor. It would be interesting to explore scenarios where more applications may be supported by the network and the nodes. Also, having support for different applications with the same duty cycle would be interesting to verify. As such, further experiments are envisaged using nodes randomly deployed to identify shortcomings.

- **IPv6 Multicast Support**: By the time we started our developments, ContikiOS did not yet implemented IPv6 multicast. For that reason, we had to design our *Application-Driven Multicast Mechanism*. Since last ContikiOS version released (3.0) IPv6 Multicast is supported; it would be interesting to use this implementation which could open new research topics.

- **ZigBee IPv6-based stack for 802.15.4 networks**: Since ZigBee was not able to easily plug its kind of networks into the IP-based Internet, the ZigBee alliance designed the *ZigBee IPv6-based stack* for 802.15.4 networks, which changed the *"traditional" ZigBee* stack to use *6LoWPAN*, to use *RPL* as routing protocol, and to use UDP. In this context, the deployment of both the *RPL-BMARQ* and its *Application-Driven node synchronization mechanism* in this new stack would open an interesting engineering topic.

- **IEEE 802.11ah (low power)**: Wi-Fi networks are widely available and interconnected to the Internet. Although the Wi-Fi power consumption can be an issue for WSN, the latest IEEE 802.11ah Low power Wi-Fi standard addresses this issue. Therefore, this low-power Wi-Fi standard may be emerging as an alternate to IEEE 802.15.4. The deployment of both the *RPL-BMARQ* and its *Application-Driven node synchronization mechanism* in this new standard arises as another interesting research topic.

- **Node mobility**: In this thesis node mobility was not considered. We have always assumed that the nodes in the network are static. It would be interesting to investigate how our contributions behave in a dynamic WSN, in which the nodes may move.

References

[1] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 147–163, Dec. 2002. [Online]. Available: http://doi.acm.org/10.1145/844128.844143

[2] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, ser. SenSys '03. New York, NY, USA: ACM, 2003, pp. 138–149. [Online]. Available: http://doi.acm.org/10.1145/958491.958508

[3] J. van Greunen and J. Rabaey, "Lightweight time synchronization for sensor networks," in *Proceedings of the 2Nd ACM International Conference on Wireless Sensor Networks and Applications*, ser. WSNA '03. New York, NY, USA: ACM, 2003, pp. 11–19. [Online]. Available: http://doi.acm.org/10.1145/941350.941353

[4] H. Dai and R. Han, "Tsync: A lightweight bidirectional time synchronization service for wireless sensor networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 8, no. 1, pp. 125–139, Jan. 2004. [Online]. Available: http://doi.acm.org/10.1145/980159.980173

[5] I. F. Akyildiz, M. C. Vuran, O. B. Akan, and W. Su, "Wireless sensor networks: a survey revisited," *Computer Networks Journal (ELSEVIER SCIENCE)*, 2006.

[6] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer Networks*, vol. 52, no. 12, pp. 2292 – 2330, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128608001254

[7] M. Dohler (Ed.) and T. Watteyne (Ed.) and France Telecom (R&D), "Urban WSNs Routing Requirements in Low Power and Lossy Networks," Routing Over Low power and Lossy networks (Active WG), April 2008.

[8] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645 – 1660, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X13000241

[9] IEEE-Computer-Society, "IEEE Std 802.15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)," September 2006, Revision of IEEE Std 802.15.4-2003.

[10] G. Montenegro and N. Kushalnagar and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," September 2007, IETF.

[11] C.-M. Tang, Y. Zhang, and Y.-P. Wu, "The P2P-RPL Routing Protocol Research and Implementation in Contiki Operating System," in *Instrumentation, Measurement, Computer, Communication and Control (IMCCC), 2012 Second International Conference on*, 2012, pp. 1472–1475.

[12] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC 6550 (Proposed Standard), Mar. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6550.txt

[13] B. F. Marques and M. P. Ricardo, "Application-Driven design to extend WSN lifetime," in *Proc. 1st Portuguese National Conference on Sensor Networks (CNRS2011), Coimbra, Portugal*, March 2011.

[14] Marques, B. and Ricardo, M., "Improving the energy efficiency of WSN by using application-layer topologies to constrain RPL-defined routing trees," in *Ad Hoc Networking Workshop (MED-HOC-NET), 2014 13th Annual Mediterranean*, June 2014, pp. 126–133.

[15] Bruno Marques and Manuel Ricardo, "Energy-efficient node selection in application-driven wsn," *Wireless Networks, The Journal of Mobile Communication, Computation and Information*, January 2016. [Online]. Available: http://link.springer.com/article/10.1007%2Fs11276-016-1194-2

[16] A. Dunkels, "Contiki OS, open source, highly portable, multi-tasking operating system for memory-efficient networked embedded systems and wireless sensor networks ," 2013, available at http://www.contiki-os.org.

[17] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-Level Sensor Network Simulation with COOJA," in *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, 2006, pp. 641–648.

[18] "Crossbow TelosB," available at http://www.memsic.com/userfiles/files/Datasheets/WSN/6020-0094-02_B_TELOSB.pdf.

[19] K.-J. Park, R. Zheng, and X. Liu, "Editorial: Cyber-physical systems: Milestones and research challenges," *Comput. Commun.*, vol. 36, no. 1, pp. 1–7, Dec. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2012.09.006

[20] E. Ancillotti, R. Bruno, and M. Conti, "Reliable data delivery with the ietf routing protocol for low-power and lossy networks," *Industrial Informatics, IEEE Transactions on*, vol. 10, no. 3, pp. 1864–1877, Aug 2014.

[21] V. C. Gungor and G. P. Hancke, "Industrial wireless sensor networks: Challenges, design principles, and technical approaches." *IEEE Transactions on Industrial Electronics*, vol. 56, no. 10, pp. 4258–4265, 2009. [Online]. Available: http://dblp.uni-trier.de/db/journals/tie/tie56.html#GungorH09

[22] E. Callaway, P. Gorday, L. Hester, J. Gutierrez, M. Naeve, B. Heile, and V. Bahl, "Home networking with ieee 802.15.4: a developing standard for low-rate wireless personal area networks," *Communications Magazine, IEEE*, vol. 40, no. 8, pp. 70–77, Aug 2002.

[23] D. E. Culler, "Wireless mesh networks promise low power ip-based connectivity," *Industrial Ethernet Book*, no. 49, pp. 16–22, November 2008.

[24] S. Deering and R. Hinden, "RFC 2460 Internet Protocol, Version 6 (IPv6) Specification," Internet Engineering Task Force, December 1998. [Online]. Available: http://tools.ietf.org/html/rfc2460

[25] Montenegro, Gabriel and Kushalnagar, Nandakishore and Hui, Jonathan and Culler, David, "RFC 4944 - Transmission of IPv6 Packets over IEEE 802.15.4 Networks," IETF RFC.

[26] J. Hui and D. Culler, "Extending ip to low-power, wireless personal area networks," *Internet Computing, IEEE*, vol. 12, no. 4, pp. 37–45, July-Aug. 2008.

[27] T. Narten, E. Nordmark, W. Simpson, and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)," RFC 4861 (Draft Standard), Internet Engineering Task Force, Sep. 2007, updated by RFC 5942, 6980, 7048. [Online]. Available: http://www.ietf.org/rfc/rfc4861.txt

[28] H. R. Babu and U. Dey, "Routing protocols in ipv6 enabled lowpan: A survey," *Int. J. Sci. Res. Publ. IJSRP*, vol. 4, no. 2, 2014.

[29] K. K. (Ed), S. D. P. (Ed), G. Montenegro, S. Yoo, and N. Kushalnagar, "6lowpan ad hoc on-demand distance vector routing (load)," June 2007, iETF, Network WG.

[30] IETF Network Working Group, "Ad-hoc on-demand distance vector (aodv) routing algorithm," Nokia Research Center, University of California, Santa Barbara and University of Cincinnati, available at http://tools.ietf.org/html/rfc3561.

[31] IETF, "The Internet Engineering Task Force," 2010. [Online]. Available: http://www.ietf.org/

[32] ROLL, "Routing Over Low Power and Lossy Networks," 2004. [Online]. Available: https://datatracker.ietf.org/wg/roll/charter/

[33] C. Chauvenet, B. Tourancheau, D. Genon-Catalot, P.-E. Goudet, and M. Pouillot, "A communication stack over plc for multi physical layer ipv6 networking," in *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*, Oct 2010, pp. 250–255.

[34] T. Clausen and U. Herberg, "Multipoint-to-Point and Broadcast in RPL," in *Network-Based Information Systems (NBiS), 2010 13th International Conference on*, Sept 2010, pp. 493–498.

[35] C. T. and H. U., "Comparative study of RPL-enabled optimized broadcast in Wireless Sensor Networks," in *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2010 Sixth International Conference on*, Dec 2010, pp. 7–12.

[36] J. Tripathi, J. de Oliveira, and J. Vasseur, "Applicability study of rpl with local repair in smart grid substation networks," in *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*, Oct 2010, pp. 262–267.

[37] M. Abdellatif, J. Oliveira, and M. Ricardo, "Neighbors and relative location identification using rssi in a dense wireless sensor network," in *Ad Hoc Networking Workshop (MED-HOC-NET), 2014 13th Annual Mediterranean*, June 2014, pp. 140–145.

[38] J.-P. Vasseur and A. Dunkels, *Interconnecting Smart Objects with IP: The Next Internet*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010.

[39] O. Gaddour and A. Koubâa, "{RPL} in a nutshell: A survey," *Computer Networks*, vol. 56, no. 14, pp. 3163 – 3178, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128612002423

[40] J. Martin, "Distribution of the time through a directed acyclic network," *European Journal of Oper. Res.*, vol. 13, pp. 44–66, 1965.

[41] O. Gnawali and P. Levis, "RFC 6719 - The Minimum Rank with Hysteresis Objective Function," in *Internet Engineering Task Force (IETF), Request for Comments: 6719*, September 2012.

[42] P. Thubert, "Objective Function Zero for the Routing Protocol for Low-Power and Lossy Networks (RPL)," RFC 6552 (Proposed Standard), Sep. 2011.

[43] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko, "The Trickle Algorithm," RFC 6206 (Proposed Standard), Mar. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6206.txt

[44] J. Martocci, P. D. Mil, N. Riou, and W. Vermeylen, "Building Automation Routing Requirements in Low-Power and Lossy Networks," RFC 5867 (Informational), Internet Engineering Task Force, Jun. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5867.txt

[45] A. Brandt, J. Buron, and G. Porcu, "Home Automation Routing Requirements in Low-Power and Lossy Networks," RFC 5826 (Informational), Internet Engineering Task Force, Apr. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5826.txt

[46] K. Pister, P. Thubert, S. Dwars, and T. Phinney, "Industrial Routing Requirements in Low-Power and Lossy Networks," RFC 5673 (Informational), Internet Engineering Task Force, Oct. 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5673.txt

[47] M. Dohler, T. Watteyne, T. Winter, and D. Barthel, "Routing Requirements for Urban Low-Power and Lossy Networks," RFC 5548 (Informational), Internet Engineering Task Force, May 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5548.txt

[48] S. Deering, W. Fenner, and B. Haberman, "Multicast Listener Discovery (MLD) for IPv6," RFC 2710 (Proposed Standard), Internet Engineering Task Force, Oct. 1999, updated by RFCs 3590, 3810. [Online]. Available: http://www.ietf.org/rfc/rfc2710.txt

[49] J. Vasseur, M. Kim, K. Pister, N. Dejean, and D. Barthel, "Routing Metrics Used for Path Calculation in Low-Power and Lossy Networks," RFC 6551 (Proposed Standard), Internet Engineering Task Force, Mar. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6551.txt

[50] Z. S. Organization, "Zigbee specification," January 2008, document 053474r17. [Online]. Available: http://www.zigbee.com

[51] L. Hester, Y. Huang, O. Andric, A. Allen, and P. Chen, "NeuRon netform: a self-organizing wireless sensor network," in *Proceedings of the Eleventh International Conference on Computer Communications and Networks*, 2002, pp. 364–369.

[52] N. Accettura and G. Piro, "Optimal and secure protocols in the ietf 6tisch communication stack," in *Proc. of IEEE International Symposium on Industrial Electronics (ISIE)*, Jun. 2014.

[53] D. Sturek, "ZigBee IP Stack overview," 2009.

[54] "Tmote Sky," available at http://www.snm.ethz.ch/Projects/TmoteSky.

[55] LCIS, "WiSMOTE," accessed: 11-Jan-2016. [Online]. Available: http://wismote.org

[56] Zolertia, "Z1 Mote," accessed: 11-Jan-2016. [Online]. Available: http://zolertia.com/products/z1

[57] Crossbow, "Wireless Modules, MICAz 2.4GHz," accessed: 11-Jan-2016. [Online]. Available: http://www.openautomation.net/uploadsproductos/micaz_datasheet.pdf

[58] T. Instruments, "eZ430-RF2500 Development Tool User's Guide," accessed: 11-Jan-2016. [Online]. Available: http://www.ti.com/lit/ug/slau227f/slau227f.pdf

[59] TI, "CC1110 & CC2510 Development Kit User Manual," accessed: 11-Jan-2016. [Online]. Available: http://www.ti.com/lit/ug/swru134a/swru134a.pdf

[60] M. Karani, A. Kale, and A. Kopekar, "Wireless sensor network hardware platforms and multi-channel communication protocols: A survey," in *Proceedings on 2nd National Conference on Information and Communication Technology, New York, NY, USA*, 2011, pp. 20–23.

[61] W. Dong, C. Chen, X. Liu, and J. Bu, "Providing OS Support for Wireless Sensor Networks: Challenges and Approaches," *Communications Surveys Tutorials, IEEE*, vol. 12, no. 4, pp. 519–530, Fourth 2010.

[62] TinyOS Working Group, "TinyOS," accessed: 11-Jan-2016. [Online]. Available: http://www.tinyos.net/

[63] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," *SIGPLAN Not.*, vol. 35, no. 11, pp. 93–104, Nov. 2000. [Online]. Available: http://doi.acm.org/10.1145/356989.356998

[64] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesc language: A holistic approach to networked embedded systems," in *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, ser. PLDI '03. New York, NY, USA: ACM, 2003, pp. 1–11. [Online]. Available: http://doi.acm.org/10.1145/781131.781133

[65] T. V. Chien, H. N. Chan, and T. N. Huu, "A comparative study on operating system for wireless sensor networks," in *Advanced Computer Science and Information System (ICACSIS), 2011 International Conference on*. IEEE, 2011, pp. 73–78.

[66] T. Zhang and X. Li, "Evaluating and analyzing the performance of rpl in contiki," in *Proceedings of the First International Workshop on Mobile Sensing, Computing and Communication*, ser. MSCC '14. New York, NY, USA: ACM, 2014, pp. 19–24. [Online]. Available: http://doi.acm.org/10.1145/2633675.2633678

[67] J. Hui and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks," RFC 6282 (Proposed Standard), Sep. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6282.txt

[68] J. Ko, S. Dawson-Haggerty, O. Gnawali, D. Culler, and A. Terzis, "Evaluating the Performance of RPL and 6LoWPAN in TinyOS," in *Proceedings of the Workshop on Extending the Internet to Low power and Lossy Networks*, ser. IP+SN '11, Apr. 2011.

[69] 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*, ser. SenSys '03. New York, NY, USA: ACM, 2003, pp. 126–137. [Online]. Available: http://doi.acm.org/10.1145/958491.958506

[70] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors," in *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, Nov 2004, pp. 455–462.

[71] Swedish ICT, "SICSlowpan - Internet for low-power, low-cost Wireless Project," accessed: 07-Jan-2016. [Online]. Available: https://www.sics.se/projects/sicslowpan-internet-for-low-power-low-cost-wireless

[72] A. Dunkels, "The ContikiMAC Radio Duty Cycling Protocol," Swedish Institute of Computer Science, Tech. Rep. T2011:13, Dec 2011. [Online]. Available: http://dunkels.com/adam/dunkels11contikimac.pdf

[73] N. Tsiftes, J. Eriksson, and A. Dunkels, "Low-power wireless ipv6 routing with contikirpl," in *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, ser. IPSN '10. New York, NY, USA: ACM, 2010, pp. 406–407. [Online]. Available: http://doi.acm.org/10.1145/1791212.1791277

[74] J. Ko, J. Eriksson, N. Tsiftes, S. Dawson-haggerty, A. Terzis, A. Dunkels, and D. Culler, "Contikirpl and tinyrpl: Happy together," in *In Proceedings of the workshop on Extending the Internet to Low power and Lossy Networks (IP+SN 2011*, 2011.

[75] L. Guan, K. Kuladinithi, T. Potsch, and C. Goerg, "A deeper understanding of interoperability between tinyrpl and contikirpl," in *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on*, April 2014, pp. 1–6.

[76] J. Ko, J. Eriksson, N. Tsiftes, S. Dawson-Haggerty, M. Durvy, A. Terzis, A. Dunkels, and D. Culler, "Beyond interoperability: Pushing the performance of sensornet ip stacks," in *In Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys*. Citeseer, 2011.

[77] J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, T. Voigt, R. Sauter, and P. J. Marrón, "Cooja/mspsim: Interoperability testing for wireless sensor networks," in *Proceedings of the 2Nd International Conference on Simulation Tools and Techniques*, ser. Simutools '09. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, pp. 27:1–27:7. [Online]. Available: http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5637

[78] Q. Cao, T. Abdelzaher, J. Stankovic, and T. He, "The liteos operating system: Towards unix-like abstractions for wireless sensor networks," in *Information Processing in Sensor Networks, 2008. IPSN '08. International Conference on*, April 2008, pp. 233–244.

[79] E. Baccelli, O. Hahm, M. Wählisch, M. Günes, and T. Schmidt, "RIOT: One OS to Rule Them All in the IoT," INRIA, Research Report RR-8176, Dec. 2012. [Online]. Available: https://hal.inria.fr/hal-00768685

[80] E. Baccelli, O. Hahm, M. Günes, M. Wählisch, and T. Schmidt, "Riot os: Towards an os for the internet of things," in *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, April 2013, pp. 79–80.

[81] H. Will, K. Schleiser, and J. Schiller, "A real-time kernel for wireless sensor networks employed in rescue scenarios," in *Local Computer Networks, 2009. LCN 2009. IEEE 34th Conference on*, Oct 2009, pp. 834–841.

[82] The Sensor Network Museum, "Tmote Sky," accessed: 08-Jan-2016. [Online]. Available: http://www.snm.ethz.ch/Projects/TmoteSky

[83] N. Pantazis, S. Nikolidakis, and D. Vergados, "Energy-efficient routing protocols in wireless sensor networks: A survey," *Communications Surveys Tutorials, IEEE*, vol. 15, no. 2, pp. 551–591, Second 2013.

[84] L. Catarinucci, R. Colella, G. Del Fiore, L. Mainetti, V. Mighali, L. Patrono, and M. L. Stefanizzi, "A cross-layer approach to minimize the energy consumption in wireless sensor networks," *International Journal of Distributed Sensor Networks*, vol. 2014, p. 11, 2014. [Online]. Available: http://dx.doi.org/10.1155/2014/268284%]268284

[85] I. F. Akyildiz and M. C. Vuran, *Factors Influencing WSN Design*. John Wiley and Sons, Ltd, 2010, pp. 37–51. [Online]. Available: http://dx.doi.org/10.1002/9780470515181.ch11

[86] G. Anastasi, M. Conti, M. D. Francesco, and A. Passarella, "Energy conservation in wireless sensor networks: A survey," *Ad Hoc Networks*, vol. 7, no. 3, pp. 537 – 568, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870508000954

[87] V. Raghunathan, C. Schurgers, S. Park, and M. B. Srivastava, "Energy-aware wireless microsensor networks," *IEEE Signal Processing Magazine*, vol. 19, no. 2, pp. 40–50, Mar 2002.

[88] G. J. Pottie and W. J. Kaiser, "Wireless integrated network sensors," *Commun. ACM*, vol. 43, no. 5, pp. 51–58, May 2000. [Online]. Available: http://doi.acm.org/10.1145/332833.332838

[89] A. Dunkels, F. Osterlind, N. Tsiftes, and Z. He, "Software-based on-line energy estimation for sensor nodes," in *EmNets '07: Proceedings of the 4th workshop on Embedded networked sensors*. New York, NY, USA: ACM, 2007, pp. 28–32.

[90] H.-Y. Zhou, D.-Y. Luo, Y. Gao, and D.-C. Zuo, "Modeling of node energy consumption for wireless sensor networks," *Wireless Sensor Network*, vol. 3, no. 1, p. 18, 2011.

[91] L. Alazzawi and A. Elkateeb, "Performance evaluation of the wsn routing protocols scalability," *Journal of Computer Systems, Networks, and Communications*, vol. 2008, 2008. [Online]. Available: http://dx.doi.org/10.1155/2008/481046

[92] F. Koushanfar, N. Taft, and M. Potkonjak, "Sleeping coordination for comprehensive sensing using isotonic regression and domatic partitions," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, April 2006, pp. 1–13.

[93] P. Santi, "Topology control in wireless ad hoc and sensor networks," *ACM Comput. Surv.*, vol. 37, no. 2, pp. 164–194, Jun. 2005. [Online]. Available: http://doi.acm.org/10.1145/1089733.1089736

[94] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi, "In-network aggregation techniques for wireless sensor networks: a survey," *Wireless Communications, IEEE*, vol. 14, no. 2, pp. 70–87, April 2007.

[95] I. F. Akyildiz and M. C. Vuran, *Time Synchronization*. John Wiley and Sons, Ltd, 2010, pp. 243–263. [Online]. Available: http://dx.doi.org/10.1002/9780470515181.ch11

[96] D. L. Mills, "Internet time synchronization: the network time protocol," *IEEE Transactions on Communications*, vol. 39, pp. 1482–1493, 1991.

[97] W. Su and I. Akyildiz, "Time-diffusion synchronization protocol for wireless sensor networks," *Networking, IEEE/ACM Transactions on*, vol. 13, no. 2, pp. 384–397, April 2005.

[98] T. Ma, Z. Xu, M. Hempel, D. Peng, and H. Sharif, "Performance analysis of a novel low-complexity high-precision timing synchronization method for wireless sensor networks," *Wireless Communications, IEEE Transactions on*, vol. 13, no. 9, pp. 4758–4765, Sept 2014.

[99] M. Al Ameen, S. M. R. Islam, and K. Kwak, "Energy saving mechanisms for mac protocols in wireless sensor networks," *International Journal of Distributed Sensor Networks*, vol. 2010, p. 16, 2010. [Online]. Available: http://dx.doi.org/10.1155/2010/163413%]163413

[100] M. Miller and N. Vaidya, "A mac protocol to reduce sensor network energy consumption using a wakeup radio," *Mobile Computing, IEEE Transactions on*, vol. 4, no. 3, pp. 228–242, May 2005.

[101] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient mac protocol for wireless sensor networks," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, 2002, pp. 1567–1576 vol.3.

[102] T. van Dam and K. Langendoen, "An adaptive energy-efficient mac protocol for wireless sensor networks," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, ser. SenSys '03. New York, NY, USA: ACM, 2003, pp. 171–180. [Online]. Available: http://doi.acm.org/10.1145/958491.958512

[103] W. Pak, K.-T. Cho, J. Lee, and S. Bahk, "W-mac: Supporting ultra low duty cycle in wireless sensor networks," in *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, Nov 2008, pp. 1–5.

[104] A. El-hoiydi and J. d. Decotignie, "Wisemac: an ultra low power mac protocol for the downlink of infrastructure wireless sensor networks," in *in 9th International Symposium on Computers and Communications (ISCC '04*, 2004, pp. 244–251.

[105] Y. Sun, O. Gurewitz, and D. B. Johnson, "Ri-mac: A receiver-initiated asynchronous duty cycle mac protocol for dynamic traffic loads in wireless sensor networks," in *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, ser. SenSys '08. New York, NY, USA: ACM, 2008, pp. 1–14. [Online]. Available: http://doi.acm.org/10.1145/1460412.1460414

[106] M. R. Ahmad, E. Dutkiewicz, X. Huang *et al.*, "A survey of low duty cycle mac protocols in wireless sensor networks," *InTech*, 2011.

[107] R. Maheswar, P. Jayarajan, and F. N. Sheriff, "A survey on duty cycling schemes for wireless sensor networks," *International Journal of Computer Networks and Wireless Communications*, vol. 3, no. 1, p. 37, 2013.

[108] J. Elson and D. Estrin, "Time synchronization for wireless sensor networks." in *IPDPS*, vol. 1, 2001, p. 186.

[109] M. R. H. Khan, M. A. Hossain, and M. S. H. Mukta, "Zigbee cross layer optimization and protocol stack analysis on wireless sensor network for video surveillance," in *International Conference on Electronics, Computer and Communication (ICECC 2008)*. Bangladesh: University of Rajshahi, 2008, pp. 795–799.

[110] L. D. Mendes and J. J. Rodrigues, "A survey on cross-layer solutions for wireless sensor networks," *Journal of Network and Computer Applications*, vol. 34, no. 2, pp. 523 – 534, 2011, efficient and Robust Security and Services of Wireless Mesh Networks. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804510002079

[111] Chipcon Products and Texas Instruments, "2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver," 2006, document SWRS041.

[112] IEEE Computer Society, "IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)," September 2006.

[113] JENNIC, "Calculating 802.15.4 data rates," Jennic, August 2006, application Note: JN-AN-1035.

[114] B. Latré, P. D. Mil, I. Moerman, B. Dhoedt, P. Demeester, and N. V. Dierdonck, "Throughput and Delay Analysis of Unslotted IEEE 802.15.4," *JNW*, vol. 1, no. 1, pp. 20–28, 2006.

[115] R. Jain, D.-M. Chiu, and W. Hawe, "A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems," *CoRR*, vol. cs.NI/9809099, 1998. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr9809.html#cs-NI-9809099

[116] J. Boudec, *Performance Evaluation of Computer and Communication Systems*, ser. Computer and communication sciences. EFPL Press, 2010. [Online]. Available: http://books.google.pt/books?id=nibpCdEjUEYC

[117] "BeagleBone Black," 2014, available at http://beagleboard.org/BLACK.

[118] P. Pinto, A. Pinto, and M. Ricardo, "End-to-end delay estimation using rpl metrics in wsn," in *Wireless Days (WD), 2013 IFIP*, Nov 2013, pp. 1–6.

[119] V. Paxson, M. Allman, H. J. Chu, and M. Sargent, "Computing TCP's Retransmission Timer," Internet Engineering Task Force, June 2011. [Online]. Available: http://tools.ietf.org/html/rfc6298

[120] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with cooja," in *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, 14-16 2006, pp. 641 –648.

[121] "RaspBerry Pi," 2014, available at http://www.raspberrypi.org.

[122] R. Carli, A. Chiuso, L. Schenato, and S. Zampieri, "Optimal synchronization for networks of noisy double integrators," *Automatic Control, IEEE Transactions on*, vol. 56, no. 5, pp. 1146–1152, May 2011.

[123] J. HE, P. Cheng, L. Shi, and J. Chen, "Sats: Secure average-consensus-based time synchronization in wireless sensor networks," *Signal Processing, IEEE Transactions on*, vol. 61, no. 24, pp. 6387–6400, Dec 2013.

[124] J. He, P. Cheng, L. Shi, J. Chen, and Y. Sun, "Time synchronization in wsns: A maximum-value-based consensus approach," *Automatic Control, IEEE Transactions on*, vol. 59, no. 3, pp. 660–675, March 2014.

[125] Hunter, J.S., "The Exponentially Weighted Moving Average," *Quality Technology*, vol. 18, pp. 203–210, 1986.

[126] V. Jacobson, "Congestion avoidance and control," *SIGCOMM Comput. Commun. Rev.*, vol. 18, no. 4, pp. 314–329, Aug. 1988. [Online]. Available: http://doi.acm.org/10.1145/52325.52356

Appendix A

Exponentially Weighted Moving Average

The *Exponentially Weighted Moving Average* (EWMA) [125] is a technique used for calculating a run-time average characterized by giving less and less weight to data as they get older and older. EWMA is easily plotted and may be also viewed as a forecast for the next observation. The EWMA equals the present predicted value plus *lambda* times the present observed error of prediction,

$$EWMA = \hat{y}_t + \lambda(y_t - \hat{y}_t) \tag{A.1}$$

where $\hat{y}_t$ is the predicted value at time t (the old EWMA), y_t is the observed value at time t, $y_t - \hat{y}_t$ is the observed error at time t, and λ is a constant $(0 < \lambda < 1)$ that determines the depth of memory of the EWMA. Eq. A.1 can be written as

$$\hat{y}_{t+1} = \lambda y_t + (1-\lambda)\hat{y}_t \tag{A.2}$$

EWMA statistics are currently used, for instance, by TCP to recover from undelivered segments; the mechanism is based on [126] and EWMA is used to estimate the timeout value that depends on Round Trip Time.