TIMELYWSN: MIDDLEWARE ARCHITECTURE FOR PROBABILISTIC QOS SUPPORT IN WSNs

João André Varino Alves

DISSERTAÇÃO
MESTRADO EM SEGURANÇA INFORMÁTICA

2013
TIMELYWSN: MIDDLEWARE ARCHITECTURE FOR PROBABILISTIC QOS SUPPORT IN WSNs

João André Varino Alves

DISSERTAÇÃO

MESTRADO EM SEGURANÇA INFORMÁTICA

Dissertação orientada pelo Prof. Doutor António Casimiro Ferreira da Costa e co-orientado por Luís Miguel Madeira Marques

2013
Acknowledgments

There are many people I would like to thank for the support they provided during the course of the work that lead to this dissertation, and indeed during the entire course of the Masters programme.

First, I would like to thank my supervisor, Prof. António Casimiro, for his guidance, support, and availability. I would also like to thank my co-supervisor Luís Marques, whose research this thesis is based on, and who was always available to help and contribute to my own work.

Next I would like to thank the Department of Informatics and LaSIGE which have always been excellent places to study and work.

I would be remiss not to thank my fellow Masters students, especially my colleagues of lab 25, and above all Igor Antunes and Pedro Nóbrega da Costa.

Finally I would like to thank my family, my grandmother Maria, my mother Maria, my brother João, and my girlfriend Nicole, who were always there for me, and gave me the hearth to carry on even in the worst of times.
À minha avó
Abstract

Recently, there has been a lot of interest and research in the field of wireless sensor networks (WSNs). Wireless ad-hoc networks made of hundreds to thousands of computer nodes equipped with sensory equipment, that measure the state of physical entities. WSNs can be used in numerous scenarios and applications, for instance, in forest fire prevention and detection, industrial monitoring, or patient-health monitoring. Many of these applications will have real-time requirements. However, WSNs present some unique issues, that make the provisioning of hard real-time guarantees an unreasonable objective. Hard real-time systems are deterministic and predictable, while WSNs are, by nature, unpredictable. Alternatively, some researchers have explored the provisioning of some Quality of Service (QoS), as an alternative to hard real-time guarantees.

This thesis describes the study and definition of a middleware architecture for the support of applications with real-time requirements in WSNs, built on the concepts of probabilistic QoS guarantees, statistical monitoring, and application adaptation. It identifies a set of underlying services that need to be implemented as part of this framework, explaining why they are needed and what they provide. Furthermore This thesis presents, Probabilistic Latency Estimation based Routing (PLER), a routing protocol for WSNs, that takes advantage of the middleware services, serving as a proof-of-concept application.

**Keywords:** Wireless Sensor Networks, Real-time systems, Routing protocols, Middleware
Resumo

Recentemente, tem existido muito interesse e investigação no campo das redes de sensores sem fios (WSNs), redes ad-hoc sem fios, compostas por centenas a milhares de nós computacionais, equipados com equipamento sensorial, que medem o estado de entidades físicas. As WSNs podem ser utilizadas em numerosos cenários e aplicações, por exemplo, para a prevenção e detecção de incêndios florestais, monitorização industrial, ou monitorização da saúde de pacientes. Muitas destas aplicações terão requisitos tempo-real. No entanto, as WSNs apresentam alguns desafios únicos, que tornam o aprovisionamento de garantias tempo-real estritas, um objectivo irrealista. Os sistemas tempo-real são por norma deterministas e previsíveis, enquanto que as WSNs são, por natureza, imprevisíveis. Como alternativa, alguns investigadores têm explorado o aprovisionamento de algum nível de serviço (QoS), como uma alternativa às garantias tempo-real estritas.

Esta tese descreve o estudo e definição de uma arquitetura de middleware para o suporte a aplicações com requisitos tempo-real em redes de sensores sem fios, construída sobre os conceitos de garantias probabilistas, monitorização estatística e adaptação aplicacional. São identificados um conjunto de serviços que necessitam de ser implementados como parte deste sistema, bem como a razão dessa necessidade. Para além disso, esta tese apresenta, encaminhamento baseado em estimação probabilística de latências (PLER), um protocolo de encaminhamento para WSNs, que toma vantagem dos serviços do middleware, servindo como prova de conceito.

Palavras-chave: Redes de sensores sem fios, sistemas tempo-real, protocolos de encaminhamento, middleware
Resumo Alargado

Recentemente tem havido um grande interesse e pesquisa na área das redes de sensores sem fio, redes ad-hoc compostas por entre centenas a milhares de nós computacionais equipados com equipamento sensorial, que medem o estado de entidades físicas. Equipamentos, tais como leitores de temperatura ou dispositivos de rastreamento de movimento, ligados por comunicação rádio sem fio. Uma das razões para este interesse são os avanços na tecnologia, que permitem o fabrico de nós sensores mais baratos, mais pequenos e mais poderosos. Isto torna-os ideais para a monitorização de um conjunto diversificado de ambientes, tais como florestas, fábricas, hospitais, só para citar alguns. As aplicações podem usar estas redes para obter uma representação virtual do ambiente, monitorizar, e até mesmo agir com base nas condições e eventos observados. Por exemplo, num cenário de monitorização de florestas, quando os sensores detectam um incêndio, a aplicação pode então avisar as autoridades, dando informações precisas sobre a localização e a dimensão do incêndio.

Na maioria dos casos, os nós de sensores têm um poder de computação limitado e mais importante, reservas de energia escassas, o que limita o que estes nós podem fazer. Assim, um modelo comum é a existência de uma estação central (ou sink), que agrega as leituras dos sensores e executa as operações de computação mais complexas. Esta estação central de processamento, não partilha as limitações dos nós de sensores, e pode ser considerado como sendo um sistema mais tradicional, ligado a uma rede de energia ou uma fonte de energia mais poderosa, e também ligado à Internet. Por exemplo, no cenário de monitorização de incêndios florestais, a estação central pode estar localizada no edifício do guarda florestal, recebendo dados de nós sensores espalhados pela floresta.

Um desafio neste campo é o da prestação de garantias tempo-real para aplicações em redes de sensores sem fios, e este continua a ser um problema em aberto. A visão clássica de tempo-real, exige que o sistema seja determinista e previsível, o que não é o comportamento típico, ou mesmo desejável, para uma rede de sensores sem fios.

Esta imprevisibilidade decorre de problemas, como falhas de nó ou interferências de transmissão na comunicação sem fio. Uma maneira comum de lidar com nós falhados, falhas por omissão e alguns outros problemas é empregar redundância. Por exemplo, utilizando vários caminhos ou enviando a mesma mensagem várias vezes. Este é um método testado e válido para redes tradicionais. As redes de sensores geralmente possuem grandes
quantidades de nós redundantes, no entanto têm uma limitação que precisa de ser levada em conta, e que é a natureza limitada dos seus recursos energéticos. Como as redes de sensores geralmente têm valores reduzidos de energia disponível, a sobrecarga introduzida pela redundância não pode ser negligenciada. De acordo com alguns autores, o objectivo operacional de uma rede de sensores deve ser a de satisfazer os requisitos das aplicações através do uso de recursos redundantes estritamente necessários. Simplesmente aumentar a quantidade de recursos ou de redundância só empurra a fasquia de falhas toleradas, mas não garante necessariamente que os requisitos serão cumpridos. EM vez de tentar fornecer garantias de tempo-real estrito, estes mesmos autores oferecem como alternativa o provisioningamento de garantias de nível de serviço probabilísticos. Quando uma aplicação solicita uma certa qualidade de serviço, a rede pode tentar igualar os requisitos (com uma certa probabilidade), analisando o comportamento estatístico das perturbações e adaptar-se, de modo a oferecer a quantidade adequada de recursos redundantes.

Esta tese descreve o estudo e a definição de uma arquitetura de middleware para o suporte de aplicações com requisitos de tempo real em redes de sensores sem fios, construída sobre os conceitos de garantias probabilistas, monitorização estatística e adaptação aplicacional. Além disso, descreve o projeto, implementação, implementação e avaliação de um protocolo de encaminhamento, que embora sirva principalmente para apresentar estes conceitos, foi na verdade projetado em simultâneo com a arquitetura, e ajudou a refinar a sua definição.

Os objetivos do trabalho presente foram: estudar o conceito de garantias de nível de serviço probabilistas, e chegar a uma arquitetura de apoio para aplicações com requisitos de tempo real, com base nesse conceito. Rever as opções existentes no que diz respeito a sistemas operativos e bancos de ensaio para redes de sensores sem fios (várias universidades na Europa e ao redor do mundo oferecem acesso a redes de testes), e apurar os mais adequados para o processo de desenvolvimento, implementação e avaliação do middleware e do protocolo de encaminhamento. Ganhar conhecimento e experiência com estas ferramentas de desenvolvimento e implementação de redes de sensores, úteis não só para o presente trabalho, mas para futuros projetos do grupo de pesquisa (a concretização deste objectivo permite superar o fato de que o departamento não tem uma rede de testes onde implementar e avaliar aplicações para redes de sensores). Desenvolver uma aplicação prova de conceito para o conceito de QoS probabilística, e testá-lo tanto no simulador como em ambiente de banco de ensaios, e comparar os resultados da simulação e banco de ensaios, com o objetivo de averiguar a qualidade da simulação.

As principais contribuições deste trabalho foram: o desenho de uma arquitetura de garantias probabilísticas na forma de um serviço de middleware para o suporte de aplicações com requisitos de tempo-real. Detalhando o modelo de serviço, componentes necessários, a interface aplicacional e outras informações relevantes para a utilização de middleware ou extensão do mesmo. O desenho, implementação, e avaliação de um protocolo de enca-
minhamento que integra aspectos desta arquitetura, e serve como uma prova de conceito.

Em termos de trabalho futuro, a avaliação do protocolo de encaminhamento (PLER) teve algumas falhas. Seria desejável comparar o PLER e o LQER (o protocolo base do PLER), tanto em relação ao objectivo do protocolo original (reduzir a taxa de perda), quer em termos de latência ponta-a-ponta.

Em termos de middleware, há uma série de possibilidades para explorar. Outras métricas de QoS, o uso simultâneo de várias métricas. Serviços e funcionalidades adicionais, só para mencionar alguns. Um ponto de interesse particular, e que foi, em grande parte, deixado de fora desta tese, é o equilíbrio entre os recursos energéticos e os requisitos de QoS. Desenvolver os mecanismos necessários para gerir estes recursos e exigências, de forma dinâmica, é uma questão interessante ainda por ser resolvida.

Mecanismos e as especificações dos módulos, aplicações e extensões escritas para o middleware, a fim de fornecer algum nível de interoperabilidade e padronização, também é outra perspetiva interessante. Em sistemas distribuídos tradicionais, middleware tem sido usado com sucesso para padronizar e fazer a ponte entre aplicações, escondendo a heterogeneidade de software e hardware. Mas em redes de sensores este ainda é um tema pouco desenvolvido.

Por último, uma área excitante de estudo é a fronteira entre as redes de sensores, que são, em essência redes ad-hoc, e as redes tradicionais, e como se pode desenhar um middleware que ultrapasse esse limite. Em outras palavras, descobrir uma arquitetura de middleware capaz de suportar aplicações adaptáveis com requisitos de tempo real e confiabilidade, que se estendem por cidades inteiras, e diferentes sistemas computacionais e paradigmas. Um middleware para a internet das coisas, no verdadeiro sentido da palavra.
Contents

List of Figures xix
List of Tables xxi

1 Introduction 1
  1.1 Motivation 2
  1.2 Objectives 2
  1.3 Contributions 3
  1.4 Institutional Context 3
  1.5 Publications 3
  1.6 Document structure 4

2 Related Work 5
  2.1 Wireless Sensor Networks 5
    2.1.1 Anatomy of a mote 5
    2.1.2 Communication 6
    2.1.3 Applications 6
  2.2 Real-time Systems 7
    2.2.1 Hard real-time systems 7
    2.2.2 Soft real-time systems 7
    2.2.3 Mission-critical real-time systems 7
  2.3 Routing in WSN 8
    2.3.1 Network Structure 10
    2.3.2 Protocol Operation 13
  2.4 Middleware Services 14
    2.4.1 Resource management 14
    2.4.2 Network topology 15
    2.4.3 Heterogeneity 15
    2.4.4 Real-time support 15
    2.4.5 Application knowledge 15
    2.4.6 Data aggregation 15
2.4.7 Quality of Service ........................................... 16
2.4.8 Security ..................................................... 16

2.5 Operating Systems ........................................... 16
2.5.1 OS Architecture ........................................... 17
2.5.2 Programming model ...................................... 18
2.5.3 Scheduling ................................................... 18
2.5.4 Memory management ..................................... 18
2.5.5 Communication ............................................ 19
2.5.6 Resource sharing ......................................... 19

3 Probabilistic QoS Guarantees ................................. 21
3.1 Perception quality ............................................ 22
3.2 Statistical inference ........................................ 22
3.2.1 Parametric statistics .................................... 23
3.2.2 Non-parametric statistics ............................... 23
3.3 Monitoring methodology ................................. 24
3.3.1 Sampling .................................................. 24
3.3.2 Analysis ................................................... 25
3.3.3 Adaptation ............................................... 25
3.4 Summary ....................................................... 26

4 Middleware Architecture for probabilistic QoS support in WSNs ........................................ 27
4.1 Problem definition ........................................... 27
4.2 Design principles ........................................... 28
4.2.1 General ................................................... 29
4.2.2 Weak assumptions ..................................... 29
4.2.3 Lightweight .............................................. 29
4.2.4 Extendable ............................................... 30
4.3 Assumptions .................................................. 30
4.4 Goals .......................................................... 31
4.5 Middleware boundaries ................................... 31
4.6 Middleware Architecture ................................. 35
4.6.1 Monitoring Service ..................................... 35
4.6.2 Auxiliary Services .................................... 36
4.6.3 Middleware Interface .................................. 39
4.7 Summary ....................................................... 41

5 Probabilistic Latency Estimation based Routing ........................................ 43
5.1 LQER .......................................................... 43
5.2 From LQER to PLER ....................................... 44
5.3 Implementation ................................................. 47
  5.3.1 Development environment .......................... 47
  5.3.2 Implementation ........................................... 48
5.4 Summary .......................................................... 50

6 Testing and Evaluation ........................................... 51
  6.1 Testing setup .................................................. 51
  6.2 Methodology ................................................... 52
  6.3 Results and evaluation ...................................... 53
  6.4 Summary ........................................................ 56

7 Conclusion .......................................................... 57
  7.1 Future work directions ...................................... 58

Abbreviations .......................................................... 61

Bibliography ........................................................... 66
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Normal histogram. Image reproduced from [23]</td>
<td>23</td>
</tr>
<tr>
<td>3.2</td>
<td>Cumulative histogram. Image reproduced from [23]</td>
<td>24</td>
</tr>
<tr>
<td>4.1</td>
<td>Conventional layered setup</td>
<td>31</td>
</tr>
<tr>
<td>4.2</td>
<td>Fully independent routing and middleware layers</td>
<td>32</td>
</tr>
<tr>
<td>4.3</td>
<td>Middleware with access to the routing layer</td>
<td>33</td>
</tr>
<tr>
<td>4.4</td>
<td>Reduced middleware</td>
<td>34</td>
</tr>
<tr>
<td>4.5</td>
<td>Overall view of the system</td>
<td>36</td>
</tr>
<tr>
<td>4.6</td>
<td>Monitoring component</td>
<td>37</td>
</tr>
<tr>
<td>4.7</td>
<td>Auxiliary components</td>
<td>38</td>
</tr>
<tr>
<td>6.1</td>
<td>Cooja simulation results</td>
<td>53</td>
</tr>
<tr>
<td>6.2</td>
<td>Twist results</td>
<td>54</td>
</tr>
<tr>
<td>6.3</td>
<td>Comparison of Cooja and Twist results</td>
<td>55</td>
</tr>
</tbody>
</table>
List of Tables

4.1 Partial interface for the monitoring service . . . . . . . . . . . . . . . . . . 41
4.2 Alarm interface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
Chapter 1

Introduction

Recently, there has been a lot of interest and research in the field of wireless sensor networks (WSNs), wireless ad-hoc networks made of hundreds to thousands of computer nodes equipped with sensory equipment, that measure the state of physical entities, equipment such as temperature readers or motion tracking devices, connected by wireless radio communication. One of the reasons for this interest, is the advances in the technology, which allow the manufacturing of cheaper, smaller and more powerful sensor nodes. This makes them ideal for widespread monitoring of a diverse set of environments, such as forests, industrial plants, hospitals, just to name a few. Applications can then use these networks to get a virtual representation of the environment, monitor, and even act based on the observed conditions and events. For example, in a forest monitoring scenario, when sensors detect a fire, the application can then warn the authorities, giving precise information about the location and dimension of the fire.

For the most part, wireless sensors have limited computing power and, more importantly, scarce energy reserves, which limits what these nodes can do. So a common model is to have a central base station (also called sink node), which aggregates sensor readings and executes the more complex computing operations. This central processing station, doesn’t share in the limitations of the sensor nodes, and can be thought of as a more traditional computer system, connected to a power grid or a more powerful source of energy, and also connected to the internet. For example in the forest fire monitoring scenario, the base station could be situated in the forest guard building, receiving data from sensor nodes scattered through the forest.

WSNs can present themselves in many different shapes and forms, with parameters such as network size, node density, and number of sink nodes varying according to the objective of the network, or the applications running on it. Another aspect that has an impact on the operation of a WSN is the environment where the network is deployed, since the sensor nodes might be exposed to ambient hazards (rain, heat, theft, etc.). These and
other uncertainties in WSNs, make data processing, communication, and sensor management a considerable challenge, with most solutions only applicable to a subset of WSN configurations [7].

An even greater challenge is that of providing real-time guarantees for applications running on WSNs, and this remains an open problem. The classical view of real-time, requires that systems be deterministic and predictable, which is not the typical, or even desirable, behavior for a WSN. Several authors have explored the provisioning of some Quality of Service (QoS) guarantees as an alternative to hard real-time, but as will be discussed in [3], these alternatives are not without their issues.

This thesis describes the study and definition of a middleware architecture for the support of applications with real-time requirements in WSNs, built on the concepts of probabilistic QoS guarantees, statistical monitoring, and application adaptation [23]. Furthermore it describes the design, implementation, deployment and evaluation of a routing protocol, that while serving primarily to showcase these concepts, was in fact designed alongside the architecture, and helped refine its definition.

1.1 Motivation

The present work was motivated by the following reasons:

1. The absence of a detailed model for the probabilistic guarantees concept, that offered an architectural view, a description of the various components and provided services to ease the development of WSN applications with real-time probabilistic requirements.

2. The fact that the concepts had never been evaluated on a sensor testbed (a real network of radio connected physical sensor nodes) only on software simulators.

1.2 Objectives

The objectives of the present work were:

1. To study the probabilistic Quality of Service (QoS) concept, and reach an architecture that provided support for applications with real-time requirements, based on that concept.

2. To review existing options in regards to Wireless Sensor Network (WSN) Operating Systems and testbeds (several universities in Europe and around the world offer
access to their resources), and ascertain the ones best suited for the implementation, deployment and evaluation portions of the present work.

3. To gain knowledge and experience using these development and deployment tools for WSNs, useful not only for the present work, but for future projects of the research group. Meeting this objective allows overcoming the fact that the department does not have a testbed in which to deploy and evaluate medium to large size WSNs.

4. To develop a proof-of-concept application for the probabilistic QoS concept, and test it both on simulator and testbed environment, and to compare the simulation and testbed results, with the objective of ascertaining the quality of the simulation.

1.3 Contributions

The main contributions of this thesis are:

1. The design of an architecture for probabilistic real-time guarantees in the form of a middleware service for the support of applications with real-time requirements, detailing the service model, necessary components, provided API, and other relevant information for the middleware use or extension.

2. The design, implementation, deployment and evaluation of a routing protocol which integrates aspects of this architecture, and serves as a proof-of-concept application.

1.4 Institutional Context

The present work took place at the Large-Scale Informatics Systems Laboratory (LaSIGE), a research unit of the Department of Informatics (DI) of the University of Lisbon, Faculty of Sciences. It was developed within the scope of the KARYON - Kernel-Based ARchitecture for safetY-critical cONtrol, project.

1.5 Publications

Some of the ideas and work of this thesis generated the following publication:

1.6 Document structure

This document is structured as follows:

- Chapter 2 - Literature review and related work on the basic underlying concepts (WSN, real-time, routing protocols).

- Chapter 3 - Description of the probabilistic QoS guarantees concepts, upon which the present work was developed.

- Chapter 4 - Description of the middleware architecture for probabilistic QoS support designed in the course of the present work.

- Chapter 5 - Description of the adapted routing protocol designed in the course of the present work, with the objective of testing the middleware solution, and description of the adapted routing protocol implementation process.

- Chapter 6 - Testing and evaluation of the adapted routing protocol.

- Chapter 7 - Concluding remarks and future work.
Chapter 2

Related Work

The subject of this thesis is the design and definition of a middleware architecture for probabilistic QoS support in WSNs, this chapter provides the necessary background, or basic concepts for the understanding of the different topics which make up the present work. It also presents work and literature that constitute the foundation, or is worth mentioning, in the context of the present work.

The first section of this chapter introduces the concept of wireless sensor network, its method of operation and components. The second section introduces real-time systems (RTS), the definition of real-time and its importance in WSNs. This definition is relevant in the context of this work, because as is discussed in [3] hard real-time is an unrealistic goal for WSNs, and this constitutes one of the motivations of much of the work presented in this thesis. The third covered topic, is an essential aspect of WSNs, routing. A thorough overview is provided, because the practical component of this thesis, which is used for testing and functions as a proof-of-concept application, is in fact a routing protocol. The fourth section of this chapter discusses the concept of middleware, the chosen approach to probabilistic QoS support for this thesis. The last section covers the topic of operating systems (OSs), and discusses two of the most popular offerings for WSNs.

2.1 Wireless Sensor Networks

The field of Wireless Sensor Networks is one that has seen a great deal of interest in the past few years, and is at the intersection of three different research areas [7]: sensing, communication and computing, benefiting from the fast development of all these areas.

2.1.1 Anatomy of a mote

A wireless sensor node or mote is the usual designation for the devices that make up a WSN. Motes (term coined by researchers at Berkeley) can have different configurations, ranging from coin shaped boards with less than 1cm to the more standard 5 cm squared
boxes [19]. A mote is composed of several elements: microcontroller, transceiver, memory, power source, and one or more sensors that can be internal onboard, or external, plugged through ports. [19] provides a comparison of various WSN mote technologies, comparing their capabilities, sensor support, and price, among other characteristics.

2.1.2 Communication

WSNs have a lot in common with Ad-Hoc Networks and glsMANET. For instance, they both require self-organization capabilities and use hop-to-hop communication. WSNs have a fairly simple protocol stack for communication, physical layer, Medium Access Control (MAC) layer, and networking layer. A brief description of each layer is presented next:

Physical layer

The standard physical layer technology used in WSNs is radio frequency transceivers (although infrared or optical media can also be used [3]), which have the purpose of translating bits into radio signals and broadcasting them.

MAC layer

The MAC layer is responsible for setting up communication links between nodes (point-to-point i.e. hop-to-hop) for data transfer, and for the sharing of communication resources between nodes.

Network layer

The MAC layer allows nodes to communicate with their one-hop neighbors, the network layer allows, through the use of multi-hop routing protocols, for a node to communicate with virtually any other node in the network. The present work focuses on this layer, and as such, section 2.3 provides further details on this topic.

2.1.3 Applications

WSN have a wide range of applications. Typical examples are temperature and other weather related phenomena monitoring or environmental monitoring, such as forest fire detection. They can also be used for security and counterterrorism, using a network of video and audio sensors to provide authorities with better means to fight these threats [7]. Another example is the use of sensor networks in industrial settings, where they can be used to monitor "machine" health and provide a picture of a factory environment, scanning and predicting for malfunctions and sounding alarms, thus keeping the workplace safer.
One final example is any sort of application for which traditional wired sensor systems are in place, that would benefit from a change to wireless communication.

2.2 Real-time Systems

[30] defines a Real Time System (RTS) as “a system whose progression is specified in terms of timeliness requirements dictated by the environment”, another definition, by [20] is that a real-time system is a system where “the correctness of a computation is defined both in terms of the logical results and the time at which they are provided”, in simple terms, results are correct and are delivered on time (i.e. within some pre-determined time bound). An aspect of these systems that cannot be overlooked is the notion that timeliness requires synchrony [30], and so the principles and techniques used to build synchronous systems are of special relevance to RTS (an example is the need for clock synchronization techniques).

Real-time systems can be divided into different classes, with different constraints on the timeliness requirements:

2.2.1 Hard real-time systems

In these systems, timing failures (failure to meet a deadline) cannot occur, that is, once the deadline is violated, the computation is considered incorrect.

2.2.2 Soft real-time systems

In soft real-time systems, while timing failures should still be avoided, they are accepted, and if in hard RTS, the utility (correctness) of a computation turns to 0 once the deadline is missed, by contrast in soft RTS the utility of a computation progressively decreases towards 0 once the deadline is missed [8].

2.2.3 Mission-critical real-time systems

In mission-critical real-time systems, timing failures should be avoided, but can still occur. The difference to soft real-time systems, is that when timing failures occur, the system is prepared to handle them as exceptional events, thus mitigating the consequences on provided service, in order to ensure that the mission is successfully accomplished.

The concept of determinism plays a central role in real-time systems and in what class they belong. Contrary to some misconceptions, real-time is not about performance and speed, nor can it be achieved simply by having a “faster” system. Chapter [3] discusses the relation between determinism and the impossibility of hard-real time on WSN, suffice it
to say that for a system to be hard real-time, it requires complete determinism.

In the context of WSNs, real-time is becoming increasingly important as the need for lower latency, timely communication arises [21]. Applications in health-care, critical-infrastructure monitoring and target tracking, are instances where outdated information is irrelevant and even prejudicial, and prime examples of scenarios that require real-time support.

2.3 Routing in WSN

Routing in Wireless Sensor Networks is a particularly challenging problem when compared to routing in wired or even other wireless networks such as cellular or ad-hoc networks [4]. The challenge arises from the particular characteristics of these networks, such as scale and the limited resources of the hardware. The scale, possible thousands of nodes, makes the overhead of maintaining a global addressing scheme too costly, so protocols should strive to maintain only local information (for example next hop nodes), and stay clear of IP like addressing. Another issue of WSNs that has a deep impact on the routing protocols, is the need for self-organization. Since nodes are deployed in an ad-hoc manner (without prior knowledge of where each individual node will be located in relation to the base-station and other nodes), once the nodes start operating they need to establish connections with other nodes and somehow form routes to the base-station. Finally, the biggest challenge is perhaps the constrained nature of resources in WSNs, meaning that every possible solution to the other problems must be efficient and lightweight, in order to preserve the lifetime of the network [2].

There are several routing issues and concerns that must be addressed when designing a routing protocol for WSNs [4] [2], depending on the objectives of the network and the applications using it some might be more relevant than others, and many are in fact contradictory (in the sense that to address one concern some other might have to be sacrificed), so it is important to know these issues and to properly balance them. These are some of those issues:

Node Deployment

Node deployment in WSNs can be deterministic or randomized. With deterministic deployment nodes are placed deliberately (manually for example, or in a grid) and packets can be routed through predetermined paths. With randomized node deployment, nodes are scattered randomly, which can lead to situations where some nodes are disconnected from the network, poorly connected, or where partitions can occur.
Node Heterogeneity

Nodes in WSNs are usually (at least assumed to be) homogeneous (all nodes have the same hardware specifications), but this might not be true. There might be different types of nodes (for example, some older then others) and the types of instrumentation associated to each node might also be different, for example some nodes might have cameras, while others might have temperature readers. These differences might have impacts, for example on data traffic.

Data Reporting

There are several models for data reporting according to the needs of the applications. Data reporting can be time-driven, event-driven, query-driven or hybrid. The time-driven model is used for applications that require periodic data, that is sent at regular intervals. The event-driven model is suitable for aperiodic data, where the sensor sends data in response to an event, for example when a fire-detection system detects smoke. In the query-driven model it’s the application that sends a request for data.

Energy Consumption

One of the most important aspects of wireless sensor nodes, is that for the most part they have a limited supply of energy, usually provided by a battery. This means that every computation and every transmission consumes some amount of power, so to prolong the lifetime of the sensor network it is essential to reduce these operations to a minimum, which requires efficient protocols.

Scalability

Sensor networks can have anywhere from a few dozen to thousands of nodes, so routing protocols should scale properly. Any sort of protocol that uses global information such as global unique identifiers or routing tables is unfit for this sort of network. Rather, protocols should aim at relying on local information, usually about nodes that are within radio distance. These nodes are usually referred to as neighbor nodes, or one-hop nodes.

Fault Tolerance

Wireless sensor nodes operate in open, harsh environments, so node equipment and link failures should be expected and accounted for. The network should adapt either by finding new routes, or have some degree of redundancy already in action.
Quality of Service

Certain applications have need of special requirements from the network, usually associated with the end-to-end latency of message delivery. We call these requirements Quality of Service. In an unpredictable environment such as WSNs, guaranteeing QoS is especially problematic.

There is a large body of work on routing protocols that try to deal with some or all of the above issues, and two ways of categorizing these routing protocols are identified in [4]: According to their network structure and according to their protocol operation. When categorizing according to network structure, relevance is given to the way the nodes are connected or structured. Examples of categories are: flat based routing, where all the nodes behave the same (have the same “job”), and hierarchical based routing, where nodes have different routing responsibilities. When categorizing according to protocol operation, relevance is given to the way the protocol works. An example of a category is multipath based routing, where multiple routes from source to sink must be found in order to provide some measure of fault tolerance. There is also a third way to categorize routing protocols, which is according to the route discovery process. Here the categories are reactive (or on-demand), when routes are created when needed, proactive (or table-driven), when routes are created before hand so that they are already established when needed, and hybrid, which makes use of both reactive and proactive techniques.

The following section reviews the different categories of routing protocols, from both the network structure and the protocol operation perspectives, identifying the most relevant abstract notions for each category.

2.3.1 Network Structure

Routing protocols in WSNs can be divided into categories according to network structure. Usually they are divided into three categories: flat-based, hierarchical-based and location-based routing. In flat-based routing all nodes have the same tasks and roles, while in hierarchical-based routing there are different classes of nodes, each with different tasks and responsibilities. Location-based routing is characterized by nodes exploiting topological and position knowledge to route data.

Flat Routing

In flat routing, nodes in the network are generally similar and perform the same set of tasks. The communication is multi-hop and usually directed towards or from a base-station. Because of the overhead associated with maintaining global unique ID’s, flat-based routing protocols are usually data-centric, meaning that the base-station dissemi-
nates interest or queries through the network instead of contacting a specific node, when a node receives a query, if it has the requested data, it forwards it to the base-station. Below are described some of the more relevant flat routing abstractions.

**Flooding** The wireless medium has a broadcast nature, which means that unlike a wired network, where the nature of the medium is point to point (a link between two computers, hubs, switches, etc.), in the wireless medium data is not sent to a single destination, but to anyone within range of the radio signal. Network flooding takes advantage of this fact, to forward a message from a source to a destination, using a simple algorithm [15]. When a node receives a message it re-broadcasts it. This way, eventually, all nodes in the network receive the message, ensuring that the destination node also receives it. But basic flooding has some serious problems. If unchecked it can lead to a broadcast storm, a phenomenon where the network is flooded with messages to the point where it ceases to operate in a correct way. Another issue of flooding is that if too many nodes are trying to broadcast a message within range of each other, it will lead to a lot of collisions (i.e. lost messages), requiring re-transmissions, which is detrimental to the performance of the nodes and the network in general. Much of the work done in the design of routing protocols for wireless sensor networks has been to find ways of improving upon basic flooding.

**Gossip** Gossip is another well known routing technique [15]. It is a variation of flooding, where upon receiving a message, a node re-transmits the message with a probability $p$ and discards the message with a probability $1-p$. By properly adjusting the probability $p$, one can avoid the pitfalls of flooding at the cost of some delay on the propagation of the message through the network.

**Minimum Cost Forwarding Algorithm (MCFA)** The MCFA [34], is an algorithm for WSNs where all communication is directed towards a sink or base-station. Because of this characteristic, it is very simple to create optimal paths between every node and the sink. It is also a scalable algorithm, since all the information a node needs to maintain is the minimum cost path from itself to the base station. The algorithm is briefly explained below.

The first step is for the base station to broadcast a message to start the minimum cost path discovery. This message can be sent any time the base station wants to update the paths. When a node receives one of these messages from a neighbor it checks to see if that neighbor is on its minimum cost path, and if it is, the node adds its own cost and broadcasts the message, otherwise it doesn’t. Eventually the message will make its way through all the nodes on the network and the minimum cost paths from all nodes to the sink will be established. When a node wants to send a message it broadcasts it with the cost
from itself to the base station, this way a node that receives the message can check to see if it is on the shortest (minimum cost) path to the base station. If it is, then it broadcasts the message, and so on until it reaches the sink.

**Directed Diffusion**  Directed diffusion [18], is a data-centric paradigm in the sense that all data is characterized by an attribute-value pair. When the base station wants information, it propagates an interest through the network, which is a task it wants done. While the data is being propagated, gradients towards the sink are being set up, which are then used to “draw” the data matching the interest. With directed diffusion, nodes also try to aggregate data on its way to the sink. Directed diffusion is considered an important milestone in the data-centric routing research for WSNs [2] and has spawned many routing protocols.

**Hierarchical Routing**

Hierarchical or cluster-based routing is another model of network structure for wireless sensor networks. In hierarchical routing, instead of all nodes performing the exact same tasks, nodes are divided into clusters, with a node or subset of nodes acting as cluster-head. The cluster-head has the responsibility of aggregating data from within the cluster and sending it to the base-station. The type of tasks performed by the cluster-head varies in complexity depending on the protocol in question, but for the most part the objective of the hierarchical model is to lower the energy consumption of nodes within a cluster (since they only need to communicate with the cluster-head, which should be in a very close radio range, and usually a node needs to spend more power to transmit to longer distances). This is also a good way to improve scalability, since the number of messages sent to the base station is reduced.

**Location based Routing**

In location based routing nodes are addressed according to their location. To know their location, nodes can either be equipped with a GPS, or a cheaper alternative, use distance estimate algorithms, based on signal strength. Once the nodes in the network know their position in relation to the base station and their neighbors they can forward traffic more efficiently (for example, instead of broadcasting a received message, a node first checks if it is closer to the destination then the node it received the message from. If it is then it broadcasts the message, otherwise it discards the message since it is not on the path to the destination node).
2.3.2 Protocol Operation

In the previous section routing protocols were reviewed according to their network structure, but another important aspect of these protocols is their operation, the way they work. The categories from this perspective are more closely related to the goals of the applications running on the network, and the protocol that is chosen to be implemented or designed should reflect their needs. It is also worth mentioning that there is an overlap between the categories based on network structure and on protocol operation. Many protocols will fall under categories of both perspectives.

In this section several categories of protocol operation for routing protocols in WSNs are described.

Query based routing

In query based routing, the base-station or sink disseminate queries through the nodes in the network, and if a node has data that matches the query it sends it to the base station, usually using some information contained in the query to guide its routing decision. An example of this sort of routing is directed diffusion, which sets up gradients towards the sink node as the query is being propagated.

Multipath based routing

Multipath based routing refers to protocols that use more then one route between source and sink nodes. This is usually done for one of two reasons. To enhance network performance or to achieve fault tolerance. However, maintaining multiple paths between nodes and the base-station has an associated overhead, so it needs to be carefully considered. Protocols will usually try to use algorithms that create disjoint paths (paths that do not share nodes) between source and sink so that the fault of a node can’t destroy more than one path. Multipath routing can also be used to prolong network lifetime, by keeping multiple choices of paths and using the one with the most "residual energy".

Negotiation based routing

The purpose of negotiation-based routing is to solve some of the problems of flooding by reducing the number of redundant transmissions in the network. To achieve this, high level descriptors or meta-data are used for the negotiation before the actual data is transmitted.

QoS based routing

In QoS based routing, the network is required to provide certain QoS metrics when delivering data to the base-station, for example, delay or bandwidth. Routing protocols
under this category, are necessary for some applications to work as intended. For example, applications that require data to be delivered in a timely manner might require from the underlying network some QoS guarantees such as latency and delay. However, strict guarantees are hard to provide in WSNs, and so the provisioning of QoS has been somewhat of a hot topic recently in WSN research.

2.4 Middleware Services

Middleware is used in traditional computer and distributed systems as a way of providing services to applications and application developers that make it easier for them to build and reuse software. Middleware can provide many functions towards this goal, be it by hiding distribution, i.e. creating the illusion that a system with distributed components is a singular entity, by hiding heterogeneity, of hardware components or operating systems for example, by providing high-level abstractions, interfaces, and standards that facilitate interoperability, or by providing a set of services that perform general purpose operations.

While middleware has been widely adopted in traditional systems, in WSNs the same cannot be said [25]. For the most part applications run directly on top of the hardware, using OS functions, programmed using the specific, sometimes custom, programming language for that OS, and compiled to run on a specific hardware configuration. [13], [25], [32] and [12], provide overviews of the challenges and key issues of middleware in WSNs. Some of these issues are related to problems specific to developing and maintaining WSNs while others are services that should be provided. The rest of this section covers these issues.

2.4.1 Resource management

What constitutes the major benefits of WSNs, small form factor, cost, and deployment versatility, are also the cause of most of the challenges associated with developing for these networks. One such challenge is the fact that resources, namely energy, are very limited, and since these networks can be made up of thousands of nodes, scattered in hostile environments, the replacement of batteries and other maintenance, for that matter, is usually not desirable or even impossible (it is a desirable trait that a WSN complete its mission in an autonomous manner), so to accommodate this requirement, middleware designed for these networks should strive for efficiency. This means, an efficient use of the processor and memory, and low-power communication.
2.4.2 Network topology

There are two main issues related to network topology: scalability and changes in the topology. Middleware for WSNs should be ready for any network size, ranging from a couple dozen, to hundreds or even thousands of nodes, without a noticeable effect on network performance as the network size grows. Middleware should also be ready to deal or adapt to changes in the network topology. Nodes in WSNs, because of the environments they are deployed in, the hardware itself, and the normal bugs and malfunctions associated with any computer or electronic system, are prone to failure, and so middleware should strive to increase WSN dependability and fault tolerance.

2.4.3 Heterogeneity

Middleware should hide the different hardware and operating systems in a network and even serve as a bridge between networks. Finding appropriate ways of connecting WSNs to more powerful traditional distributed systems has long been one of the goals of the research community, but that vision of an “internet of things”, of city spanning interconnected sensor networks is still a ways off in the future. However, middleware is certainly an important tool in achieving this goal.

2.4.4 Real-time support

Many WSN applications deal with time and space, and are of a real-time nature (meaning it is possible to assign a bound on message end-to-end latency for a given event). Middleware should thus provide real-time services to applications.

2.4.5 Application knowledge

Usually middleware strives to be as general as possible, that is, to support as many applications as possible. However, because of the nature of WSNs, middleware must allow for the injection of application knowledge into the infrastructure and WSN.

2.4.6 Data aggregation

There is usually a lot of redundancy in data in WSNs (if a lot of nodes are in close proximity they might sense the same event), so in-network data aggregation is a desirable service, which allows for the reduction of redundant transmissions, achieving as a side-effect considerable energy savings (radio communication is more “expensive” than computations).
2.4.7 Quality of Service

Quality of service can mean a lot of things. In WSNs, QoS can be divided into two categories: application specific QoS, which is directly related to parameters of an application, for example quality of the measurements, or number of active sensors, and network QoS, which is related to how the network is meeting application demands using its resources (energy, bandwidth, etc.). Traditional QoS mechanisms used in wired networks aren’t adequate for WSN, because of its wireless nature and resource limitations, therefore middleware should provide new metrics, based on a trade-off between performance and energy efficiency.

2.4.8 Security

Security has become a major concern in WSN research both for public and military applications [24], as this technology is increasingly used in sensitive scenarios (e.g. healthcare, critical infrastructure monitoring). Security in WSNs is a big challenge, due to the fact that the sensor nodes can be scattered in open areas. This makes them easy targets, and the wireless communication means that they are easy to eavesdrop on or even to inject packets into the network. The other fundamental problem, is that many of the security solutions for traditional systems are simply too expensive in terms of size and computational complexity for use in WSNs. [14] provides a study on WSN security attacks and vulnerabilities.

Comparing the issues of developing routing protocols for WSN presented in section 2.3, with the ones above, it becomes apparent that there is some overlap, a clear sign that middleware could help with the development of these protocols.

2.5 Operating Systems

The Operating System (OS) is an important component of any computer system. It is responsible for managing the systems resources, which in a traditional system means things like processors, the different kinds of memories, keyboard and other devices, network interfaces, etc. It acts as the “middleman” between the hardware and the user applications, and developers can use various OS services through system calls (OS services are different from middleware services, because the first are usually simpler and closer to the hardware, while the second usually are more complex and in user space, operating on top of the OS and using its system calls).

Operating systems for WSNs have a similar job to their traditional systems counterparts with some peculiarities associated with the resource constrained nature of WSNs.
[10], the most recent survey on WSN operating system, details the issues related to OS design in WSN. The next few sections discuss some of these issues.

2.5.1 OS Architecture

The OS architecture or organization is perhaps its most important aspect, as it will determine what functionalities the kernel, i.e. the central component of the OS, provides and how applications can interface with the kernel to use them. In WSNs, kernel memory footprint is very important, since the size of the kernel should be as small as possible because memory is very limited. There are four basic OS architectures, monolithic, layered, microkernel, and virtual-machine:

**Monolithic**

The monolithic architecture was the architecture of the first operating systems [11], and it has in fact no structure, only a bundle of independent services with interfaces between them. It has the advantage that interactions between services have low costs, and that it can be compiled into images with a small footprint [10], but it is hard to debug, understand and maintain.

**Layered**

As the name implies, in a layered architecture the OS is organized into layers, each layer implements a functionality or set of functionalities that the above layers can access through a well defined interface. The advantages of this architecture are that it is easy to understand and manage. However it lacks flexibility, and it can lead to some inefficient interactions between services (since they can’t directly interact with each other unless they are in adjacent layers).

**Microkernel**

The microkernel architecture shares some of the objectives of the layered architecture, and mitigates some of the problems of the monolithic architecture to increase system safety and robustness. The concept is simple, strip away functionalities from the kernel until it is left with a bare minimum, basic set of services. The rest of the services are implemented outside the kernel, in user-space servers, making for a simpler, smaller kernel. The main disadvantage of the microkernel architecture is poor performance, because of the frequent user to kernel context switch. However, since the number of context switches in a WSN application is far smaller, the microkernel architecture is a common design choice for WSN OS architectures [10].
Virtual-machine

A virtual machine is a piece of software that can run applications while making them believe that they are executing on top of a real machine, emulating the system [11], its advantage is portability, but the downside is performance, which makes it a poor choice for WSNs.

An OS architecture for WSNs should respect two key principles. Small memory footprint, because of the resource constraint, and partially for the same reason, flexibility (only modules required by the applications are loaded).

2.5.2 Programming model

The programming model supported by an operating system will fundamentally determine how applications are developed for it. The two most popular models for WSNs are multithreaded programming and event-driven programming. Multithreading is the most common model in traditional systems, and so the most familiar to developers. However, it is resource intensive and not really appropriate for WSN application programming. The event-driven model is a lot closer to the way a sensor network application logically functions, and is considered to be well suited for resource constrained devices [10]. However, it is a model most programmers are not familiar with, and sometimes find difficult to grasp. Some research has been done on lightweight multithreading [35] to conciliate these two realities, on one hand, the convenience of a familiar programming model, and on the other hand an efficient use of resources.

2.5.3 Scheduling

The choice of scheduling algorithm will depend on the nature of the WSN environment, particularly, if it is real-time or not. Other than that, any scheduling algorithm for WSNs should be memory and energy efficient.

2.5.4 Memory management

The memory management strategy is the way the OS deals with the allocation and deallocation of memory for processes and threads, while trying to ensure that there are no overlaps (one process writing on the address space of another) or leaks. Memory management can be static or dynamic [11]. While static memory management is useful for sensor devices because of its simplicity, it is inflexible because no run-time memory allocation can occur. Dynamic memory allocation does not suffer from this problem, but requires careful management and thus is more complex. In the early WSN operating systems, it was assumed that nodes would only run one application, one process, so no
memory protection was needed, but since then that reality has changed considerably, so newer systems need further complex mechanisms [10].

2.5.5 Communication

In terms of communication, a WSN OS must provide inter-process communication, but beyond that it must also provide a communication Application Programming Interface (API) that allows applications to communicate with each other [10]. This means that it must have implementations of MAC layer protocols, and at least some basic transport layer protocols such as broadcast and unicast. Some operating systems go even beyond that offering complex network and transport layer abstractions such as reliable unicast or μIPv6 [9].

2.5.6 Resource sharing

As WSNs evolve, and the number of applications and processes running on a node increases, it is essential to have a proper resource sharing mechanism (especially if dealing with a multithreaded model). Resource sharing mechanisms for WSNs are no different than those of traditional systems, they can act in time (scheduling of a process on the CPU) or space (writing in system memory), and require some of the same primitives.
Chapter 3

Probabilistic QoS Guarantees

This chapter describes the concept of probabilistic QoS guarantees and other associated concepts, the fundamental building blocks of the present work and from which it spawned.

Achieving real-time guarantees in WSNs is a difficult problem, with one of the foremost issues being the lack of predictability. This unpredictability stems from problems such as node failure or transmission interference on wireless communication. One common way of dealing with faulty nodes, omission faults and some other problems is to employ redundancy. For example using multiple paths or sending the same message several times. This is a tried and valid method for wired networks, and WSNs do usually possess great amounts of redundant nodes, but they do have one limitation that needs to be taken into account, which is the limited nature of their energy resources. Because WSNs usually have a fixed, sometimes small amount of available power, the overhead introduced by redundancy cannot be overlooked. According to [23] “the operational objective of sensor networks should be that of satisfying application requirements through the use of the strictly necessary redundant resources.” The authors state that simply increasing the amount of resources or redundancy only pushes the bounds on tolerated faults, but doesn’t necessarily guarantee that the requirements will be met. Instead of trying to provide real-time guarantees, which can prove to be an impossible task, the authors offer the provisioning of probabilistic QoS guarantees as an alternative. When an application requests a certain quality of service, the network can try to match the requirements (with a certain probability) by analyzing the statistical behavior of perturbations and adapting, so as to offer the adequate amount of redundant resources.

The next section explains the concept of perception quality.
3.1 Perception quality

But when talking about QoS in WSNs it is necessary to properly define its meaning. This need arises from the fact that the term “Quality of Service” has different meanings depending on the technology and perspective it is being applied to [22]. In computer networks, QoS usually refers to latency, jitter and packet loss, but other definitions can be found. For instance, to the applications community, QoS is the quality as perceived by the user/application [22]. For the purposes of the present work, the most suitable definition is the one found in [23], which considers that for applications running on WSNs, Quality of Service is in fact data quality, i.e that the data is accurate or close to the real value of the monitored entity. [23] refers to this as a requirement for “perception quality”, in essence, how accurately the application perceives the reality. However, this high level concept of QoS, needs to be translated into a network level, observable metric. In [23], end-to-end latency is considered to be the that metric. So the lower the latency, the higher the perception quality. An example that shows how end-to-end latency relates to perception quality is for instance a movement tracking application. The longer the end-to-end latency between sensors and base-station, the worse the perception quality, because by the time the sensor data arrives at the base-station, it will no longer be an adequate representation of the real world entity that was being tracked.

3.2 Statistical inference

As stated earlier, the idea behind probabilistic QoS guarantees, is to have a network that adapts to the needs of the application. But for the network to adapt, it is first necessary to characterize its state. If we consider that every relevant QoS metric is represented by a random variable, this can be done by sampling the behavior of the network, which allows the estimation of the probability distribution of the random variables. This is called statistical inference, the process by which one draws conclusions about a parameter one is seeking to estimate, which is traditionally done using parametric methods, methods that assume that the probability distribution of the data is a well known distribution. However, these methods are computationally expensive, which makes them unsuitable for sensor nodes (that have limited computing power). In [23], the authors propose instead to achieve lightweight dependable adaptation through the use of non-parametric statistics. Non-parametric methods are distribution free, robust and lightweight, since they only require simple computations. The next sections review the concepts of parametric statistics and non-parametric statistics.
3.2.1 Parametric statistics

In parametric statistics it is assumed that the sample data corresponds to some well known family of probabilistic distributions, and they are called parametric because they have parameters that control the properties of the distribution (such as average value or dispersion) [23]. An example of a well known distribution is the normal distribution, or the Poisson distribution. Once that assumption is made then one can make very precise estimates and work with other statistical tools that can be applied to the study of the distribution. The downside to parametric statistics, is that if the assumption made about the family of distributions is wrong, then they can be very misleading. This type of statistics is not very robust because it is tightly tied to its assumptions.

Using statistical adaptation can be a way to somewhat mitigate this shortcoming. The idea is to check, using run-time statistical diagnostics, if the distribution of the samples matches any well known distribution. For instance, consider the histogram on figure 3.1 which is divided into several bins, each counting the number of sampled values that fell in that particular interval, which can then be matched to well-known distributions in search of a good fit. This is however a very computationally expensive task, because it requires not only to continually run the diagnostics, but also requires that the samples be matched against a large amount of possible distributions, these aspects make this technique unsuited for use in WSNs.

3.2.2 Non-parametric statistics

Unlike parametric statistics, non-parametric statistics are distribution free, in other words, it is not assumed that the data belongs to any given probability distribution. This makes them robust and simple, and while these methods are less accurate than parametric ones, they are non the less effective, and in a resource constrained setting such as WSNs, the fact that non-parametric methods are much more lightweight, makes up for the loss of
Figure 3.2 shows a cumulative histogram, where each bin counts the number of occurrences that fall within that bin and the preceding ones. Using this non-parametric method, it is possible to see what percentage of sample values fall below any given value. For instance, what percentage of end-to-end samples fall below the 15 ms mark.

The next sections provide insights on how non-parametric methods can be used to monitor and infer the state of the network.

3.3 Monitoring methodology

The objective of network monitoring is to infer the state of the system, with the objective of adapting, in order to provide certain QoS requirements, or to raise awareness of the network operating conditions, so that applications can adapt based on that information. The methodology described in [23] for adaptation consists of a three step cycle, first, sampling the environment, then, inferring the state of the network (i.e. analyzing the sampled data), and finally, adapting accordingly.

3.3.1 Sampling

In order to understand or infer the state of the network, and subsequently make estimations and predictions about its future behavior, it is first necessary to gather sample values of the desired QoS attribute, be it end-to-end latency (perhaps the most obvious one), or some other. Sampling can be done in several different ways, and with different sample sizes, depending on the objectives and limitations of the network or application, and will have a big impact on the accuracy of the estimations. As with many other aspects of WSNs, there is a trade-off to be considered between the expended resources and the sampling
techniques utilized. This topic will be touched upon in later topics, particularly in the design chapter, and in the implementation chapter.

3.3.2 Analysis

Analysis using non-parametric methods is actually quite simple. In [23], the authors use the beta distribution to account for sampling errors and reach an estimate value of how much of the population really is less to or equal than a sample value, and instead of doing this for the value itself, it is done for the numbers ordinal ranking on the sample. What this means is that a value is considered to cover 100% of the population not because of its value, but because it was the \( n^{th} \) order statistic of a sample with \( n \) numbers [23]. The mean of the beta distribution is given by:

\[
\text{mean} = \frac{\alpha}{\alpha + \beta} \quad \text{for} \quad \alpha = k \quad \text{and} \quad \beta = n + 1 - k.
\]

And for the \( k^{th} \) order statistic:

\[
\text{mean} = \frac{k}{n+1}
\]

Using this function it is possible to get a better understanding of the impact that the sampling and the sample size have on the estimates. For a sample size of 10 the result of the function is \( \frac{10}{11} = 0.91 \) or 91% , not 100%. For a sample size of 20, the result is \( \frac{20}{21} = 0.95 \), closer, but still does not cover the entire population. As the number of sampled values grows so to does the maximum coverage, i.e. the percentage of messages with lower end-to-end latency then the estimates. Ideally one would just keep adding samples and the coverage would increase thus allowing for better estimates, but there are trade offs that must be considered, like the necessary memory. It is impractical to use a lot of memory, to keep a large amount of samples, especially in a resource constricted system. Another aspect to be considered is the staleness of the samples, because if we are dealing with a dynamic environment, one where the attributes of the network change (arguably the most useful scenario for the use of these concepts), then older samples might not reflect the current state of the network and will just harm the monitoring process.

3.3.3 Adaptation

Adaptation comes in two flavors, namely network adaptation and application adaptation. In the first, the network (node infrastructure), adapts to meet the application demands, without the application realizing what is really happening in the background. For example the nodes might increase signal strength or the retransmission ratio, to try accommodating the application. In the latter, it is the application that adapts to the reality of the network.
state, potentially changing its method of operation or some parameters. For instance, an application with timeliness requirements might buffer sensor readings before sending them in an effort to save on transmissions (but only up to the point where it knows that no deadlines of the buffered readings will be missed), but if it is made aware that the network latency is increasing, then it might decrease the time it waits before sending the readings, knowing that they might miss the deadlines. Some of the adaptation mechanisms employed might be common to the two types. One of the reasons is the fact that WSN applications are traditionally very tightly coupled with the network infrastructure they are running on, and usually there is only one application running on each node (although this trend is changing).

3.4 Summary

This chapter described in some detail the theory behind the probabilistic QoS model. It presented key concepts such as perception quality, and goes over the necessary topics of statistics necessary to understand the present work. Finally it describes the monitoring methodology which is used in practice.

The following chapters describe the original work and contributions of this thesis. The next chapter presents the proposed architecture for probabilistic QoS support in WSNs.
Chapter 4

Middleware Architecture for probabilistic QoS support in WSNs

This chapter introduces the middleware design, underlying assumptions, goals and architecture, which constitute the approach of the present work to probabilistic QoS in WSNs. In other words, what is the problem at hand, how is it handled by the chosen approach and how the solution works.

4.1 Problem definition

It has been previously discussed in this thesis how the “particular nature” of WSNs creates a new set of issues and exacerbates some of the problems inherent to traditional distributed and networked systems. Issues such as fault tolerance, data quality, or security, which benefit from decades of research for these systems, present new challenges when applied to WSNs. This thesis focuses primarily on the problem of timeliness, in other words, of real-time in WSNs, and in simpler terms, on whether or not messages reach their destination before their allotted deadline.

However, achieving hard real-time properties in wireless sensor networks is an unrealistic objective [27], this stems from several reasons, some already discussed, but the main reason is unpredictability. Unpredictability of the environment, unpredictability of the hardware, and while it can be argued that these issues are common to all computer systems, the very purpose of WSNs accentuates them, thus making the comparison somewhat unreasonable. It must be kept in mind that WSNs come in forms and sizes, that is precisely one of their positive aspects, the flexibility, ease of deployment, self-organization. It must also be remembered, that while it is common (even throughout this thesis) to think of the typical WSN configuration as a group of static motes, connected by hop-by-hop radio communication, with one or more sink nodes, perhaps with a neighbor discovery phase at the beginning of their mission, that WSNs are ad-hoc networks and in
some cases MANETs. This presents more unpredictability. At any moment new nodes might join or leave the network, the very network, might itself not be bounded to a set of coordinates, but mobile as well. Consider for instance a group of interconnected cars, the geography and environment characteristics changing as they drive along, in this scenario, there are forms of unpredictability, and interactions, that make the provisioning of real-time properties challenging to say the least.

Instead of focusing on hard-real time properties, the present work follows directly from the research on probabilistic QoS and dependable adaptation, focusing on the problems of perception quality and subsequently network monitoring. The focus is then less about the provisioning of real-time properties then it is about raising awareness about the state of the network, in other words information on certain metrics, so that applications can adequately adapt. The question is then how to provide a correct representation of the network state? As was described in the previous chapter, the way the network state representation is constructed is using statistical inference and network monitoring, so the problems that must be addressed are how to:

**sample the network** In a way that captures the dynamics of the network and allows for proper analysis and estimation. If the sampling or monitoring of the network is done in such a way that it fails to represent the flows and dynamics of the network then the results might be misleading. This as to do with some of the aspects discussed in the previous chapter, such as the size of the sample window, and other aspects such as the rate at which values are collected or even the way they are captured.

**Analyze the sample** And produce estimates which match the reality of the network. For instance, and considering the values discussed in the previous chapter, with a sample window of size 20, for a 95% estimated coverage, which means that there is a 95% probability that a message arrives its destination with an end-to-end latency bellow the estimated upper bound value, it is expected that on average this will hold true, that is, that the estimates match the reality.

This then is the problem. In the next section the design principles of the solution are presented, followed by the underlying assumptions, the goals, and finally by the solution itself.

### 4.2 Design principles

The desired solution for the above problems should be an architecture that encapsulates the various services that provide solutions for those problems, and other associated issues,
into a coherent framework, with a well defined interface for applications to use, while at
the same time respecting the constraints of wireless sensor networks, and that as much as
possible preserves the best aspects of these networks. What this description really entails
is a middleware service.

A middleware service, as was discussed in chapter 2, can mean a lot of things, for ex-
ample a layer of software that sits between the operating system and the applications (or
between any two given components), and provides services to the latter using functions
of the former. It can perform several objectives such as hiding distribution or OS hetero-
gegeneity, providing communication abstractions, or providing real-time support. However
in WSNs there are some issues that make the use and design of middleware not as straight-
forward as in traditional systems. This section discusses some of the design principles of
the middleware solution presented in this work.

4.2.1 General

Generality is a common trait of middleware services, it is usually one of their benefits
and focal points, and in WSNs it is no different. It is expected and desirable that a mid-
dleware service cater to a broad range of applications, but since applications are usually
so integrated with a specific network and routing configuration it is necessary that the
middleware allow the integration of application knowledge. What this means is that there
must be a trade-off between what is application specific and what is general. This can
be done in different ways, for instance, through the use of embedded data that allows the
middleware to adjust its operation or by allowing direct access to the network by applica-
tions.

4.2.2 Weak assumptions

An important factor in determining if a real-time time solution is applicable or not to
a given scenario, are the underlying assumptions, and if the scenario meets them or not. Real-time solutions usually make strong assumptions about environment and system
properties that must hold true for the duration of the execution. This as was previously
discussed, is hard to guarantee in an WSN environment, thus limiting the applicability
of these types of solutions. This is then another reason to focus instead on probabilistic
QoS support, because it allows for weaker, more realistic assumptions and subsequently
a wider applicability.

4.2.3 Lightweight

One of the problems that is most associated with available middleware solutions in WSNs
is the increase in overhead. However, because of the resource constraints associated with
WSNs, it is desirable that the overhead be kept to a minimum. This means using techniques that require few additional messages, such as piggybacking monitoring data on routing messages, localized algorithms, and lightweight algorithms, both in memory and complexity.

### 4.2.4 Extendable

This design principle relates to the first one, in the sense that it too is concerned with the range of applicability. What this means is that while end-to-end latency is the quality of service metric considered in the present work, there is no reason why others could not also be taken into consideration. If the idea behind the middleware and the probabilistic QoS support concept is to raise awareness to the state of the network, then there are other metrics which might in some situations be useful, whether as auxiliary metrics, to reach more reliable results for the “main” QoS metric being monitored, for instance the level of conformity\(^1\) or as alternative QoS metrics, for instance loss rate.

These are the main principles for the middleware, which provide a guideline for its design, the next section discusses the underlying assumptions.

### 4.3 Assumptions

As was stated in the design principles, because of the nature of WSNs and the desire to have a broad applicability, the middleware design should make as few assumptions about the environment and the system as possible, and even then it should only make realistic or weak assumptions.

Nodes can fail, messages might not arrive, there are no assumptions made about the dependability of the underlying network, however it is assumed that the behavior of the environment is stochastic. This means that it is non-deterministic, and that the properties of the system are defined by random variables, as was described in section 3.2. It is also assumed that the environment has limited dynamics, in other words, that it does not change too rapidly in relation to the perception capabilities of the system. The results in [23] validate this assumption, showing that adaptation can be done which is very close to the theoretical perfect adaptation. This implies that the environment dynamics are limited, otherwise this match would not occur, since the bound values would be very different by the time the adaptation was complete. These are fairly weak assumptions thus respecting the design principles of the middleware.

\(^1\)represents the conformity between the estimates and the real end-to-end values, or in other words, it allows to understand what the is the deviation of the estimates, i.e. how far they are from the real values, whether by excess or by fault
4.4 Goals

The general goal of the middleware is to provide probabilistic QoS support for adaptive WSN applications while respecting the design principles and the assumptions. To achieve this goal it must solve the problems identified in section 4.1, it must have mechanisms to sample the network and to analyze those samples, it will also need to solve some associated issues, which are not the main goal of the middleware, but are nonetheless necessary for its fulfillment. It needs to be lightweight and support a broad range of applications. In Summary the goals of the middleware are:

1. To provide probabilistic QoS estimates based on monitoring and statistical inference techniques;
2. To raise awareness about the state of the network;
3. To support adaptation.

In the previous sections several issues were discussed that influence the design of the middleware solution. In the next few sections the middleware architecture and design decisions will be presented and discussed.

4.5 Middleware boundaries

The first design decision that had to be made was where to place the middleware layer, understanding the dynamics of the overall mote system, defining the boundaries of the middleware. Conventionally the middleware would operate above the network layer. However, one particular problematic aspect inherent to WSNs, is that the routing protocols and applications are usually tightly coupled which complicates the decision.

There are a few alternatives we can consider, which provide more or less control to the middleware and that require more or less application and network knowledge.
The first alternative is to have a stack as the one in figure 4.1. This represents the overall system, which is the one closest to what a conventional protocol stack looks like. In this setup, the middleware is hiding the layers below itself from the application layer, which means that it will need to provide a routing interface for the application layer, and not only that, it will also need a mechanism to interact with the routing protocols. And this is where it is necessary to be mindful of application knowledge issues. If the middleware is supposed to support a broad range of applications, which as it stands, is one of the design principles, then it will need to support the different routing requirements of those applications. Some might wish to send their data to a central base-station, others might function as an ad-hoc network where all nodes can communicate with each other, some might benefit from cluster-based routing, others from directed-diffusion, so either the middleware is prepared to use a wide range of routing protocols, or a single protocol that is capable of supporting the routing needs of many different types of applications, perhaps at the expense of efficiency.

One way of introducing application knowledge into this setup would be through embedded data. The application could communicate to the middleware its desired routing protocol or routing protocol abstraction, akin to what some conventional middleware services allow.

The advantage of this approach is that it requires less from the application developer, because in fact it is providing communication services and probabilistic QoS support. The disadvantage of this approach is that it requires a larger more complex middleware, and strains a bit from the purpose of the middleware, which is to provide probabilistic QoS support and awareness.

A second alternative is shown in figure 4.2. Here the middleware and routing layers are side-by-side, and the application layer seats on top of them both, using the middleware for the probabilistic QoS support services and the routing layer for communication, hence the red arrow between application and middleware, signifying that there is no routing re-
quest on that interface. Unlike the previous setup, here the middleware does not know the specifics of routing algorithms. However, it requires some way of sampling network latency. It does either by intercepting the messages from the MAC layer, or by sending its own messages in order to collect data. This however raises an issue, which is the quality of the samples. Intercepting the messages from the MAC layer, will only provide latency data for next hop neighbors, so it is not possible to make estimates about destinations that are not one hop away. And sending its own messages, using a custom routing protocol, or periodic floodings, will also yield unreliable results, since the route taken to the destination might be drastically different then the one that the routing protocol will use.

The advantages of this approach are, that it simplifies the middleware, removing the communication service aspect which was necessary in the first approach, and that it allows for the tight coupling of application and routing layers, allowing for more efficient use of the network resources. The main disadvantage of this approach is that the middleware may yield unreliable estimates.

The third approach, presented in figure 4.3 is similar to the previous one, but allows the middleware to utilize the routing layer for network monitoring purposes. This allows the middleware to gather samples of end-to-end latency using the same approximate routes that the application data will follow. This helps improves the accuracy of the estimates. Looking at this setup, it might seem that it is similar to the first approach, but the fact that applications do not use the middleware for routing purposes, means that it can stay general, only requiring simple knowledge of the routing layer, and that applications can still be as tightly integrated with the routing protocol as their needs require.

The fourth and final approach, presented in figure 4.4 is very similar to the setup in 4.3. However, there is an important distinction. Here the middleware does not explicitly use the routing layer to gather samples, instead it is the routing layer, the routing protocol, which provides the values. This allows the middleware to be completely oblivious to the details of the routing protocol, it only requires an interface which allows for the input
of end-to-end latency values. Since the boundary between application layer and routing layer are themselves a bit murky in WSNs, it is reasonable to consider that it could be the application to input the values as provided by the routing protocol. This would eliminate interactions between middleware and routing layer. There is however a small caveat connected with this fact. In many WSN operating systems the routing protocol stack is a set of abstractions built on top of each other just like any other application, so while in this approach the middleware does not have access to the higher level protocols, the ones built on top of the abstractions provided by the OS, it has access to the lower level construct such as broadcast and single-hop unicast, in this approach we simply consider that the middleware does not use them to sample data because of its overhead and reduced accuracy in the results.

The advantage of this approach is that it makes for a very simple middleware, with well defined boundaries and services. It also makes it possible for the routing protocol to benefit from some of these services, in a way acting as an application from the point of view of the middleware. But perhaps the most important advantage is that it allows for efficient sample gathering with low overhead. Since the application or routing protocols are the ones that best know the details of their operation, they are the ones capable of providing the sample values in the most efficient manner. The only downside of this setup is that the simplification of the middleware must be compensated with an increase in complexity in the routing protocol or in the application.

This section discussed the middleware boundaries, in essence establishing with which other layers it interacts. Several setups were presented, each are valid approaches with their own merits, but the design choice for the present work was the fourth approach presented in figure 4.4. The reasons for this decision were the following:

1. It accomplishes the goals of the middleware while respecting the design principles;

2. In contrast with the the approach in figure 4.1 it provides probabilistic QoS support to applications without removing their freedom and flexibility to choose and
optimize their use of routing protocols;

3. In theory it is capable of achieving more accurate estimates then the approach in figure 4.2 with lower overhead then the approach in figure 4.3.

4. It is the more “focused” of all the approaches, encapsulating the services it is suppose to offer, without offering or doing any redundant services or operations of other services (specifically the communication services).

Furthermore, looking at figure 4.4 it is possible to see that any of the other approaches, barring the one in figure 4.2 can be achieved using this one as a building block. Making this the more flexible of the four solutions.

The next section will elaborate on the middleware architecture, its components and services, and where they fit in the overall system view, following the approach of figure 4.4.

4.6 Middleware Architecture

This section will present the middleware architecture, starting with an overall view of the system components, and then “zooming” in on each component, providing a detailed description. The design follows directly from the discussion present in the various sections of this chapter.

Figure 4.5 shows the overall view of the system, divided into three major components: application, middleware, and middleware interface, and their interactions. No routing component is shown, because, as was expressed in the previous section, routing protocols can be though of as applications from the point of view of the middleware, since they have access to the exact same interface. Figure 4.5 also shows that the middleware is divided into two components, the monitoring service component, and an “auxiliary services” component, which provides support services that fall out of the domain of the monitoring service, plus an interface between the two, that allows the monitoring service to use the auxiliary services. Similarly, the middleware interface is composed of two distinct interfaces, one for the monitoring service, and another for the auxiliary services. It should be noted that the internal interface and the external interfaces, the ones accessed by the applications, are not the same and provide access to different functions and services.

4.6.1 Monitoring Service

The main component is the monitoring service. This service is responsible for monitoring system QoS metrics, such as end-to-end latency latency, node connectivity, node energy,
Figure 4.5: Overall view of the system

or packet loss. The monitoring service itself is designed as a set of extendable components, shown in figure 4.6 one for each monitored metric, and each one is responsible for handling the data provided and requests made by the application, related to that metric.

Each of these QoS metric components is also responsible for maintaining a data structure of sample values, organized by monitored node. In keeping with the approach chosen in section 4.5, the responsibility of providing a list of relevant nodes to monitor falls on the client. This can be done in several different ways, depending on the scenario: the application can provide the entire list upfront, or during the operation. In the second case this is done automatically, by adding a new node if the provided sample is from a yet unknown address. The data structures vary according to monitored metric, for instance the end-to-end latency data structure is a simple circular buffer of size equal to the desired sample size (e.g. n=20) as per discussed in section 3.3.2. When the buffer is filled, it starts replacing old values with new values.

Each QoS metric component has its own interface, as is shown in figure 4.6, providing access to its functions to the application. Each of these components can access auxiliary services through the internal interface, depending on their needs.

### 4.6.2 Auxiliary Services

The auxiliary service components are just what the name implies, a set of components which act as a toolbox for the monitoring service, the applications, and other services that might require them. Figure 4.7 details the auxiliary services, they range from services indispensable to the functioning of the middleware, such as the statistical analysis component, to support services, such as clock synchronization, to services which simply act as wrappers to those of the operating system. Other services that might be part of this toolbox are for instance, neighborhood maintenance, dynamic detection of the set of nodes...
which are within radio range of the node running the middleware, or energy expenditure analysis, useful when one of the QoS metrics is energy. As figure 4.7 also shows, some of these services might be available only to the middleware, to the application or available to both and each-other.

Again, the idea is for this toolbox to be extendable and flexible, providing helpful services to the middleware, depending on the environment, and the available resources.

The two services which are most closely related to the end-to-end monitoring service and to the goals of the middleware are the statistical analysis service, and to a lesser degree the clock synchronization service.

**Statistical analysis service**

The statistical analysis service provides estimates on the end-to-end latency upper bound, and is in fact very lightweight, both in terms of memory and complexity. It performs the analysis discussed in section 3.3.2, but instead of solving for the coverage (the mean in section 3.3.2), it solves the equation for the value of \( k \), which corresponds to the index in the ordered array of samples.

\[
k = C \times (n + 1)
\]

Since it is the monitoring component that holds the array of values, it only queries the statistical analysis service, providing the size of the sample and the desired coverage.
The statistical analysis service then returns the index, falling to the monitoring service the task of ordering the array of samples and retrieving the estimate, using the provided index value. The manner by which the monitoring component does the ordering is not very relevant, and is basically an implementation choice as to what ordering algorithm to use and when to do it. If the monitoring service is expected to be queried more often then it is provided with values, then the array should be ordered at insertion time. Because there are some algorithms that can retrieve certain values (for example the largest, or $k^{th}$ largest) from an unordered sample, it might be more efficient to do it at retrieval time.

The statistical analysis algorithm, presented in algorithm 1, is almost exactly the same as the one in [23], with the difference that the ordered sample array is not provided as input and the returned value is the index and not the estimate.

The algorithm performs the calculation of the above equation while solving some associated problems, one of them being the fact that the derived rank might fall out of bounds, if the sample size does not allow for low enough or high enough coverage. Lines 3-7 deal with this issue. The other problem is the fact that the equation might produce a non integer rank. For the most part this problem is handled seamlessly by rounding it to the nearest integer, and the border cases are handled again by lines 3-7. The algorithm also adjusts the rank for a zero-based indexing of the sample array.
Algorithm 1 Upper bound estimation algorithm

1: **Input:** Size of array sample
2: **Input:** Desired average coverage \( C \) (floating point)
3: **Output:** The index for the estimated upper bound
4: assert \((C \leq 0.0 \text{ and } C \geq 1.0)\)
5: \( \text{index} \leftarrow \lfloor (C \times (n + 1) - 0.5) \rfloor \)
6: **if** \( \text{index} < 0 \) **then**
7: \( \text{index} \leftarrow 0 \)
8: **else if** \( \text{index} \geq n \) **then**
9: \( \text{index} \leftarrow n - 1 \)
10: **end if**
11: **Return** \( \text{index} \)

Clock synchronization service

Clock synchronization is fundamental for distributed monitoring of time intervals or, in this case, of communication latencies. Because clocks naturally tend to drift and fall out of sync, without clock synchronization, not only would the measurements be off, but as time passed the errors would increase as the clocks drifted further and further apart.

The clock synchronization service is an abstraction, which the other services in the middleware, and the client itself can use, hiding the actual algorithm or mechanism that performs the synchronization. The clock synchronization service is a prime example of how a service can adapt to the available resources. The clock synchronization can be performed using a well known algorithm, such as the ones presented in [29], but if other resources are available, such as a GPS, which can improve the accuracy and precision of the clocks, then those resources should be utilized. Another example of such “out-of-band” mechanisms are large radio transmitters, with powerful signals that cover the entire WSN. Since all nodes will hear the transmission at roughly the same time, they can use this “pulse” to synchronize their clocks. Again, which clock synchronization mechanism should be used will depend on the needs of the application and the available resources.

The previous sections detailed the major components of the middleware, the next section describes the basic interface of some of those components.

4.6.3 Middleware Interface

The middleware interface is what allows the application to access the services of the monitoring component and of the auxiliary components. This section focuses on the monitoring component API, particularly for the end-to-end latency bound estimation service, the relevant QoS metric in the present work.
End-to-end latency estimation service API

The end-to-end latency estimation service API offers several functions designed to raise awareness as to the network QoS, and support application adaptation. Interface 4.1 shows some of the fundamental functions of the monitoring component. The interface reflects some of the issues that have already been discussed, for instance, an application can request that a node be monitored, either explicitly by calling monitorNode, or implicitly by simply inserting a value for a node which is not being monitored, using insertValue. Other functions, such as isMonitored, are simply useful convenience functions.

getEstimate is the function which has been most discussed up to this point, and uses the result of \[ c = \text{coverage} \] with a value \( c \) to search the sample array of node \( n \) and retrieve the estimate value. But it is also possible to have a function that does the reverse. For a chosen destination node \( n \), and a provided value of end-to-end latency \( l \), return the probability that a message will reach \( n \) before \( l \). This function is specified in interface 4.1 and requires the monitoring service to first find the index in the ordered samples array of node \( n \) where \( l \) would be inserted, then passing this value to the statistics analysis service, which will then solve the original equation from section 3.3.2, returning \( c \) which is the probability. This function only requires that the statistics analysis service be able to run a second algorithm, which should not be very different then algorithm 1.

These two simple functions provide the basic service of the monitoring component. For instance, using the first function, a sensor node that is sending temperature measurements to a sink node can become aware of the latency bound that will hold with a certain probability, say, 98%. If this latency is too high (which can imply that the sink may sometimes have a temporally inconsistent view of the temperature), then maybe the node will decide to increase the frequency of updates in order to avoid such temporal inconsistencies with the indicated probability (98%). The second function is the complement of the first. For example, the same temperature sensor node may want to know how probable it is that the latency to the sink will be, at most, 200ms. If the probability happens to be very low, then this may trigger some reconfiguration of adaptation of the node behavior in order to achieve a more predictable behavior.

These two are the functions that directly follow from the monitoring methodology presented in [23]. However, using them as building blocks, other constructs can be built. For example an alarm function as shown in interface 4.2 can be made. Using this function an application could, for instance, do the following: first call \( l = \text{getLatency} \text{(base\_station, 95)} \), which returns the estimated end-to-end latency to the base station for a probability of 95%, and then call \( \text{setAlarm} \text{(base\_station, L, 85)} \), requesting to be notified when the probability drops below 85%. The middleware will notify the ap-
Chapter 4. Middleware Architecture for probabilistic QoS support in WSNs

### Partial interface of the monitoring service

<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td><code>monitorNode(Node n, int size)</code></td>
<td>Creates a new entry for node <code>n</code> in the monitored nodes list, with a sample array of size <code>size</code>.</td>
</tr>
<tr>
<td>boolean</td>
<td><code>isMonitored(Node n)</code></td>
<td>Returns True if <code>n</code> is in the monitored nodes list, otherwise, returns False.</td>
</tr>
<tr>
<td>void</td>
<td><code>insertValue(Node n, int latency)</code></td>
<td>Inserts input value <code>latency</code> into the sample array of monitored node <code>n</code>. If <code>n</code> is not being monitored, it creates an entry in the monitored nodes for <code>n</code>.</td>
</tr>
<tr>
<td>int</td>
<td><code>getEstimate(Node n, float coverage)</code></td>
<td>Returns the end-to-end latency upper bound estimate to destination node <code>n</code> for the requested coverage percentage <code>coverage</code>.</td>
</tr>
<tr>
<td>float</td>
<td><code>getCoverage(Node n, int bound)</code></td>
<td>Returns the estimated probability that a message reaches node <code>n</code> within the provided latency bound <code>bound</code>.</td>
</tr>
</tbody>
</table>

**Table 4.1: Partial interface for the monitoring service**

<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td><code>setAlarm(Node n, int latency, float coverage)</code></td>
<td>Sets an alarm that will go off if the probability that a message will arrive at node <code>n</code> before <code>latency</code> falls below <code>coverage</code>.</td>
</tr>
</tbody>
</table>

**Table 4.2: Alarm interface**

Application if this happens, possibly triggering some sort of adaptation from the application.

### 4.7 Summary

This chapter introduced the middleware, discussing the problems it aims to solve, the design principles it should be built on, assumptions made, goals that should be achieved, and finally presents the middleware architecture itself, examining the design choices and rationale. The architecture is designed to be extendable, and to take into consideration the limitations of WSNs, providing probabilistic QoS support, while at the same time being as unobtrusive as possible to the normal developing process and positive aspects of WSNs.

While this chapter is about the middleware architecture, discussing design decisions
and possibilities, and presenting guidelines for future development and extension, the next chapter presents an example application, a proof-of-concept of sorts, which uses elements of the middleware architecture (a simplified interface and set of services), that will allow the study of some of the properties of the theoretical model for probabilistic end-to-end latency estimation as presented in [23] and discussed throughout this thesis.
Chapter 5

Probabilistic Latency Estimation based Routing

This chapter presents Probabilistic Latency Estimation based Routing (PLER), a routing protocol which performs forwarding decisions, supported by the middleware discussed in the previous chapter, as a proof-of-concept application, which will be used to test the theoretical model in a later chapter.

PLER is an adaptation to Link Quality Estimation based Routing (LQER) [6], which was chosen primarily for two reasons. First, it belongs to the family of the minimum cost forwarding algorithms (MCFA), described in section 2.3.1. These algorithms are for the most part simple and straightforward to develop and implement, making them ideal for the testing of particular, protocol independent properties, without requiring the effort of some of the more complex families of protocols. The second reason, which might seem a bit at odds with the first, is that LQER, while being a member of the MCFA family, benefits from refinements on the basic concept, from several other protocols before it, such as Minimum Hop Field based Routing (MHFR) [36]. These refinements and the ones introduced by LQER provide better opportunities for the use of the support middleware.

5.1 LQER

The LQER protocol starts with the minimum hop field establishment, which has the goal of setting up the optimal path (minimum hop-count) to send data to the sink, for each node. In this stage the sink broadcasts an ADV (advertisement) message which contains the hop count to the sink (0 at the sink). This message will be propagated through the network using the flooding algorithm and when a node $n$ receives an ADV message from a node $m$, it will compare its hop count ($h_n$) to the advertised hop count of node $m$ ($h_m$). If $h_m + 1$ is smaller than $h_n$, then $h_n$ is set to $h_m + 1$ and $n$ broadcasts the ADV message with hop count equal to $h_n$. If $h_m + 1$ is equal to $h_n$ then $n$ adds $m$ to its forwarding
table but does not broadcast the ADV message. If $h_m + 1$ is bigger than $h_n$, then $n$ simply ignores the message. At the end of this stage each node should be able to calculate the minimum hop count to the sink and have a forwarding node set, which corresponds to all nodes within radio range which are one hop closer to the sink. Algorithm 2 describes these steps.

**Algorithm 2 Minimum hop field establishment algorithm**

1: node N receives a message
2: if received hop count $< \text{current hop count}$ then
3: \hspace{1em} current hop count $\leftarrow$ received hop count + 1
4: \hspace{1em} add sender to the forwarding node set
5: \hspace{1em} broadcast current hop count
6: else if received hop count $=$ current hop count then
7: \hspace{1em} add sender to the forwarding node set
8: else
9: \hspace{1em} ignore received message
10: end if

Once the minimum hop field for each node has been established, nodes can start routing messages to the sink. When a node needs to send a message to the sink it will choose the node from its forwarding table with the best link quality, which is the node that has the largest value of $\frac{m}{k}$ where $m$ is the number of successful transmissions among the last $k$ transmissions in the sliding window for that link. A node knows that a message was successfully transmitted if it receives an Acknowledgment (ACK) message from the node it sent the message to. After the message is forwarded, the sliding window is updated to account for the success or failure of that transmission.

**Algorithm 3 Link quality sliding window maintenance algorithm, sliding window is represented by a bit-sequence**

1: transmit data packet
2: update sliding window information
3: if packet successfully transmitted then
4: \hspace{1em} shift sliding window bits left and add 1 to the rightmost
5: else
6: \hspace{1em} shift sliding bits left and add 0 to the rightmost
7: end if

### 5.2 From LQER to PLER

Using the probabilistic QoS support middleware it is possible to modify the above protocol to operate in such a way that instead of considering transmission success as its link
Algorithm 4 Link quality estimation routing algorithm

1: receive message
2: send ACK to sender
3: list all the nodes in the forwarding set
4: chose the node in the list with the largest value of \( \frac{m}{k} \)
5: send message to chosen node

quality measure, it considers end-to-end latency. That is, instead of routing messages on the link with the least loss of messages, it routes messages through the link with the best QoS, in this case meaning, the least estimated end-to-end latency to the sink node. To achieve this transition from LQER to PLER changes must be made at several stages of the protocol, particularly to algorithms 3 and 4.

The changes to 3 actually simplify the work done by the protocol, since it shifts the responsibility of maintaining the sliding window, from the algorithm to the middleware. This is because the middleware already has a sliding window of sampled latencies, making it unnecessary for the routing protocol to keep and maintain such a data structure. What the protocol must do, is feed values to the middleware, as discussed in 4.6, using function \( \text{insertValue}(\text{Node } n, \text{ int latency}) \) found in table 4.1.

This raises the question of how the routing protocol acquires those values to begin with. Ideally the solution does not stray far from the principles of the middleware, i.e. the monitoring should be done with as little overhead as possible. What that means in this case, is minimizing the cost\(^1\) of the modifications. Since, as it has been previously stated, the most costly operation is data transmission, PLER should avoid, as much as possible, to send any new messages. The way PLER achieves this, is by using a “piggyback” mechanism, and taking advantage of the broadcast nature of the wireless medium. But to understand these mechanisms it is important to first look at the message contents:

- **Next-hop node**: The ID or address of the next-hop node, i.e. the node chosen to forward the message.

- **Original sender**: The ID or address of the node where this message originated.

- **Sequence number**: The sequence number of the message, which uniquely identifies it among the set of messages of the original sender node.

- **Original timestamp**: The time at which the message was originally generated and sent, obtained using the clock service of the middleware.

\(^1\)The amount of resources expended, which in WSNs means energy
• **One-hop timestamp**: The time at which the message was last sent, obtained using the clock service of the middleware.

• **Data**: The application/sensor data.

With PLER, when a node receives a message, it sends an ACK message back to the sender, same as with LQER, except in PLER that message contains extra information, i.e. the information piggybacks on the ACK back to the sender. This extra information contains, a timestamp of when the message was created, and the last observed end-to-end latency value of the node to the base-station.

When a node receives an ACK message, it timestamps the arrival, and computes the one-hop latency by comparing the arrival timestamp with the timestamp present in the message, this is possible because the protocol uses the middleware’s clock synchronization service (this valued could also be obtained using round trip techniques). This value is then added to the observed end-to-end latency value of the sender node, thus creating a sample which it then hands over to the middleware, which will insert it into the sliding window. The node also uses this value to update the last observed end-to-end latency value to the sink, which it uses in ACK messages. This way, by propagating, and adding up latency values, starting at the sink, all the way to the edge nodes, it is possible to get end-to-end latency samples, using only localized algorithms, and up to this point, without any message overhead in relation to LQER. Obviously these end-to-end latency sample values aren’t 100% accurate, in the sense that they might not represent the end-to-end latency of a real message, but for the statistical purposes of the middleware they are a good enough approximation. The PLER equivalent to LQERs algorithm is shown in 5.

The other step of the LQER protocol that needs to be modified is the routing decision. The algorithm is shown in 6. The modifications here are simple. Instead of listing the nodes in forwarding node set and choosing the one with the best value of \( \frac{m}{k} \), PLER chooses the node with the smallest value for the desired coverage. These values are obtained calling `getEstimate(Node n, float coverage)` for each of the nodes in the node forwarding set.

One final issue must be handled, which is the fact that PLER, as it has been defined up to this point, only feeds values to the middleware concerning the node it chooses to forward its messages. The consequence, is that in practice a node will always choose the same node from the forwarding set, because the sliding windows of the other nodes will be empty. To get around this problem without the need for extra messages (for example, having all nodes send an ACK when they hear a message from a node one hop further from the base-station), PLER takes advantage of the fact that all messages in WSNs are
Algorithm 5 Sliding window maintenance

1: received ACK
2: arrival time ← getTime
3: retrieve sent time, and last observed latency from received ACK message
4: one hop latency ← arrival time - sent time
5: sampled value ← one hop latency + observed latency
6: update sliding window information
7: update observed latency to base station value

Algorithm 6 Link quality estimation routing algorithm

1: receive message
2: send ACK to sender
3: list all the nodes in the forwarding set
4: chose the node in the list with the smallest value of $getEstimate(sink, coverage)$
5: send message to chosen node

broadcasted, even the ones that are intended for a single recipient, and listens for any messages broadcasted by the nodes in its forwarding hop set. For this to be useful, a new field needs to be added to messages sent by a node, a field carrying the information about the last observed end-to-end latency to the sink, identical to the one present in ACK messages, only this field is not intended to be used by the explicit destination node of the message, but by possible nodes listening. When a node hears a message from one of the nodes in its forwarding set, it timestamps the arrival, extracts the timestamp contained in the message and the last observed latency, and hands this information over to the middleware.

The rest of this chapter will deal with the implementation process of LQER and PLER.

5.3 Implementation

This section discusses the steps taken in implementing PLER, from the choice of platform, to the implementation of LQER, and the adaptation process discussed in chapter [5]. It provides some insight into the technologies used, and issues in development.

5.3.1 Development environment

Before the protocol implementation could begin, it was necessary to choose an operating system. When programming for traditional systems, a developer can be, to a certain extent, OS agnostic, i.e. does not have to commit to run the application on a specific OS. However application programming for WSNs is usually tightly coupled to a specific OS, mainly because of two reasons: The first is that WSN OSs usually only accept a specific
programming language, and the second is that most of the functions used are OS specific as well.

The two operating systems that were considered for development purposes, were TinyOS, and Contiki-OS.

**TinyOS**

TinyOS is one of the oldest and most widely adopted WSN OS. It has a monolithic architecture, but because it only loads the components that an application requires it has a very small footprint (approximately 400 bytes). Earlier versions of TinyOS only supported the event-based programming model, but since version 2.1. multithreading is available. In terms of scheduling algorithms, it supports first-in-first-out (FIFO) and earliest deadline first (EDF).

The programming language of TinyOS is NesC, a “dialect” of the C programming language, i.e. a version of C specific for TinyOS development.

**Contiki-OS**

Contiki is a more recent OS, but has garnered a lot of support from the research community in recent years. In terms of architecture, it is modular, but does not have as small a footprint as TinyOS. It supports both the event driven and multithreading programming models. Where Contiki really shines, is in the communication protocol support. It offers IPv4 and IPv6, and a lightweight layered protocol stack called the rime stack.

Rime provides unicast, broadcast, and multi-hop communication support. Rime supports both best effort and reliable communication Rime also allows applications to run their own routing protocols.

The programming language of Contiki-OS is the C programming language.

Contiki-OS also offers a GUI network simulator called Cooja, developed alongside Contiki-OS. The quality of this simulator and its ease of use, together with the fact that Contiki has a more familiar programming model and programming language, lead to its choice as the development platform.

### 5.3.2 Implementation

Once the contiki operating system was chosen as the development environment, the task of programming the protocols began. It was decided that the implementation would be
executed in two stages:

1. In the first stage, the LQER protocol was implemented and debugged. It is a fairly straightforward protocol, and ideal to develop an understanding of Contiki’s programming model and libraries.

2. In the second stage, PLER was adapted from the code of LQER, according to section 5.2. While this stage of the work benefited from the experienced gained in the previous stage, it was also the more complex of the two, especially in terms of debugging.

**LQER implementation**

In the LQER protocol there are two types of nodes, the sink node, and sender nodes. Unlike other routing protocols, there is only one sink node, or base-station, and a node is either the sink or a sender, never both.

The code for the base-station is quite simple, it is responsible for starting the minimum hop field establishment process, and for receiving/acknowledging messages sent by other nodes. It uses the rime stack for communication, specifically the broadcast and unicast primitives. It is divided in two processes, one for each of the above communication channels and their callbacks.

The bulk of the protocol and its complexity is in the sender nodes. Again there are two processes, one for unicast and one for broadcast. The broadcast process handles the minimum hop field establishment from the sender side, while the unicast process is responsible for sending unicast messages with data, forwarding messages received from neighbors towards the base-station, and maintaining the link quality table.

**Implementation issues**

At this stage of the implementation most of the problems came from inexperience with Contiki’s libraries and programming model. However the fact that the programming language was C, contributed to a relative fast learning curve. Additionally, Contiki comes with several, well documented, example applications which for the most part are easy to follow, and help understand the different communication abstractions in the time stack and the way processes operate.

**PLER implementation**

The transformation of LQER into PLER meant a substantial increase in the complexity of the code. The increase in complexity has two origins. The first origin is the extra work
nodes have handling received messages. In the case of receiving a data message, instead of simply sending an ACK back to the sender and forwarding the message, they now have to timestamp the arrival, send the ACK with the timestamp, and forward the message. If they receive an ACK, they need to compute the one-hop latency, pass the relevant data to the middleware and update their last observed latency value.

The second and largest origin of increased complexity, is the middleware, or at least a simplified version of the middleware, implemented as a set of functions which are called by the two processes of PLER. It is responsible for handling the data provided by the protocol, and running algorithm [1]. It also provides an interface for a timestamping operation, which in this implementation is just a wrapper for Contiki’s own clock synchronization algorithm, but if other resources were available, they could be used instead.

**Implementation issues**

There was a significant increase in implementation issues and difficulties in this stage of the implementation. Unlike the previous stage, where problems where mostly due to inexperience, here the problems were in the details, and particularly it took a lot of fine tuning to get the time aspects of the protocol to operate in a correct way.

Some of the issues were also linked with the testing and evaluation, which will be discussed in the next chapter. Namely, during the testing in the physical testbed environment, there was a lot of back and forth, between adjustments to the code and retesting, because the nodes were behaving differently than what was expected, i.e. differently from the simulator used for debugging (which was supposed to emulate the real nodes).

**5.4 Summary**

This chapter presented PLER a routing protocol which bases its forwarding decisions on end-to-end latency estimations, and showcases how an application might use the services of the support middleware. This chapter also sheds some light on mechanisms that while, on this case, are being used to complement the approach shown in figure 4.4 and the architecture of section 4.6, could be applied to some of the other approaches discussed in section 4.5 through integration with the middleware.

This chapter also discussed the implementation of PLER and its base protocol LQER, as well as implementation issues. It also presented TinyOS and Contiki-OS two of the most used operating systems for WSNs. The next chapter will present the testing and evaluation of these protocols, namely, methodology, results and discussion.
Chapter 6

Testing and Evaluation

This chapter discusses the testing and evaluation of PLER specifically in regards to the probabilistic QoS concepts of chapter [3]. First the testing methodology and setup is presented, followed by the results.

6.1 Testing setup

PLER was tested on two different environments. First on the Cooja simulator, with the aim of verifying the theoretical properties of the protocol, and then on the TKN Wireless Indoor Sensor network Testbed (TWIST), of the Technical University of Berlin, with the objective of obtaining actual, real sensor network data, and comparing it with the results of the simulator. The next two paragraphs provide a brief introduction on Cooja and TWIST, and why they were chosen for testing PLER.

Cooja is the Graphical User Interface (GUI) simulator for Contiki-OS, and allows networks of all sizes to be simulated. It can also, although at reduced performance, emulate nodes at the hardware level, which allows to test applications as if they were running on actual, specific models of nodes. Cooja also has a series of features and plugins for all sorts of network related simulation and evaluation. Cooja was chosen for simulation purposes because it is the “native” simulator for Contiki, i.e. it was developed specifically and along side Contiki-OS. It is also user friendly and allows to develop and test code as if on a real testbed (many simulators require code to be written in java or another high level language, which leads to a duplication of effort when developing sensor network applications).

TWIST is a testbed developed at TU-Berlin which allows for the testing and experimenting with WSN applications. It consists of an infrastructure of over a hundred TmoteSky and eyesIFX sensor nodes scattered through three floors of one of the campus buildings. It allows for the upload of network wide code, out-of-band debug and applica-
tion data extraction, and other interesting features. It also allows access to the network, via a web interface and a Cooja plugin (for TmoteSky nodes only), to outside users for free. TWIST was chosen as the testbed for testing purposes, primarily because of its Cooja plugin and its access policy, as well as for its features.

6.2 Methodology

The objective of the PLER evaluation is to check if the protocol respects the statistical properties discussed in chapter 5. Assuming the implementation is correct, there could be two reasons why this would not be true. The first reason, could be that the assumption made about the environment’s behavior, that it is stochastic, with limited dynamics, is wrong. The second reason, could be that the manner by which the protocol collects latency samples, using the propagation method described in section 5.2, is producing inadequate values, i.e. that do not fit the theoretical model (the values collected using this method are not expected to be accurate but are expected to be adequate enough to fit the probabilistic model).

The testing of PLER is divided into two experiments, first the simulator/Cooja testing and then the tesbed/TWIST testing. The rest of this section explains how the experiments were conducted:

The objective of the experiments is to find out how close the coverage (percentage of messages that are timely) provided by PLER is to the requested coverage (the input `cov` for function `getEstimate(Base-Station, cov)`). To find the real coverage, we first associate with every message sent by a node, an estimate latency to the base-station, given by the function `getEstimate`. When a message reaches the base-station this estimate latency is compared to the actual time it took the message to arrive. The actual coverage is then the percentage of messages which have an estimate latency lower then the actual latency, i.e. for testing purposes the estimate latency is considered to be a deadline. Ideally, the actual coverage would perfectly match the requested coverage. For instance, if the requested coverage was 95% then 95% of messages would have an estimated latency lower then the actual latency, but a small margin of error is expected, especially considering the fact that the protocol is not working with perfect, accurate samples of network latency.

Each experiment consists of finding the actual coverage for several values of requested coverage, ranging from 75 to 96 percent. This is the range that is considered, in this work, to be the most useful for WSNs, since a coverage below 75 is too poor for most application requirements, and the ceiling on requested coverage for a sample window of size 30 is 96%.
The tests for the first experiment, the Cooja experiment, were run consecutively for a simulated network of 10 sender nodes and 1 base-station. The sender nodes hop-count to the base-station ranged from 1 to 3, and each node sent 200 messages, per test, to the base-station.

The tests for the second experiment were conducted in exactly the same way as the ones in the first, except the tests were run on the TWIST testbed. Again 11 nodes were used, 10 acting as sender nodes and 1 as base-station.

### 6.3 Results and evaluation

The results of the Cooja and Twist experiments are shown in figures 6.1 and 6.2 respectively.

In both figures, the diagonal straight line represents the perfect adaptation, in other words, if the requested coverage exactly matched the obtained/observed coverage. However, as was discussed in section 3.3.2, with a sample window of size n=20 or even n=30, as is the case for the experiments, the coverage is not total, i.e. the maximum coverage is not 100%. For n=20 it is 95% and for n=30 it is approximately 96.77%. Another consequence of the window size is that for some requested coverage values, the expected
coverage is actually the same. The practical result is the ladder shaped line shown in the two figures.

Because in the Cooja experiment, the tests were deterministic, identical runs, where only the requested coverage varied, the results for requested coverages with the same expected coverage were the same, hence the ladder pattern.

As for the results themselves, the observed coverage values were actually higher than the expected, especially for the lower ranges. In the higher ranges the observed values are closer to the expected coverage. Overall the results are consistent, the simulation results showing a ladder pattern similar to the expected coverage.

The results of the Twist experiment have several differences when compared to the Cooja results, as is observable in figure 6.2. The most obvious difference is that there are no observed coverage results with the same value. This is because, unlike the tests in the simulator, the runs on the testbed are not deterministic, they cannot be exactly replicated, because the testbed is a real network of hardware nodes, and not a simulation. The other difference is that there are quite a few observed coverage values that are lower than the expected coverages, especially in the higher ranges.

Figure 6.3 shows the results of the two experiments, Cooja and Twist, as well as the expected coverage line. When comparing the results of the two experiments, it is appar-
ent that PLER behaved closer to the theoretical expectations in the simulator then in the testbed. There might be several reasons for this. First, the Cooja setup is not an exact replica of the TWIST setup, i.e. nodes are not in the same positions and at the same relative distances. Second, no matter how well Cooja is able to emulate the TmoteSky nodes and replicate the network dynamics, it will still fail to capture any dynamics that are particular to the TWIST network. There are various sources of interference, such as WiFi, that have a negative impact on network performance [1], and that will affect a network in a particular way.

Still, the simulator/Cooja and testbed/Twist results are fairly similar, and in this scenario, in which Cooja is performing simulations of a generic network, without any particular knowledge or traits of the testbed, leads to the conclusion that Cooja offers a decent approximation of real networks for testing purposes during development. This however does not excuse the need for testbed testing of applications before deployment, as simulators do not and are not capable of accurately representing some network dynamics, especially those that involve external interactions, e.g. outside interference.

Finally the results of both experiments show that PLER follows the theoretical model discussed in [3] and by extension, that the proposed approach for probabilistic QoS support is working correctly. It is also an indication, that the assumption that the behavior of the environment is stochastic, is accurate, at least in the case of the TWIST testbed.
6.4 Summary

This chapter presented the experiments performed on the Cooja simulator and the TWIST testbed, to evaluate the PLER protocol and by extension the probabilistic QoS support middleware. PLER performed better in the simulated environment than in the testbed, still the results are similar, and the experiments showed that both the simulator and testbed results were consistent with what was expect, and that the Cooja simulator is a good approximation to testbed testing.
Chapter 7

Conclusion

This thesis presented an architecture for a probabilistic QoS support middleware, aimed at applications with adaptive real-time requirements in WSNs. The middleware was thoroughly discussed, the design principles, assumptions, and goals were stated, and the components that make up the middleware architecture were detailed. Additionally a routing protocol, PLER, was introduced, and its development process presented. From its inception, starting with a choice of a base protocol (LQER), then explaining the adaptations that were made, followed by the implementation phase, and finally the testing and evaluation, both on a simulator and testbed environments.

The contributions of this work comprise:

1. The design and definition of a middleware architecture for probabilistic QoS support in WSNs, including components, API, and a detailed discussion, not only on the design process, and supported services, but on the possibilities of extension, and alternative configurations.

2. PLER, a routing protocol designed to take advantage of the middleware service and model, and that at the same time, serves as a proof-of-concept application. This protocol was tested in the Cooja simulator and the TWIST testbed achieving satisfactory results.

3. The validation of the middleware assumption, that the behavior of the environment is stochastic, for a testbed setting.

During the course of this work other objectives were also met. Specifically, it allowed a building of knowledge and experience programming for the contiki-OS, working with the Cooja WSN simulator and the TWIST testbed.
7.1 Future work directions

In terms of the PLER evaluation there were some shortcomings. It would be desirable to evaluate how it compares to LQER, both in terms of the objective of the original protocol, reduce loss rate, and in terms of end-to-end latency.

In terms of the middleware, there are a lot of possibilities to explore. Other QoS metrics, the simultaneous use of multiple metrics, additional services and functionalities, just to mention a few. A particular point of interest, that was, for the most part, left out of this thesis, is to balance energy resources with QoS requirements. Developing the necessary mechanisms to manage these resources and requirements, on-demand is an interesting issue still to be addressed.

Mechanisms and specifications for modules, applications and extensions written for the middleware, in order to provide some level of interoperability and standardization, is also another interesting prospect. In traditional distributed systems, middleware has been successfully used to standardize and bridge applications, hiding software and hardware heterogeneity (for instance CORBA [31]), but in WSNs this is still an underdeveloped topic.

Lastly, an exciting field of study is the boundary between WSNs, which are in essence ad-hoc networks, and traditional networks, and how one might design a middleware that bridged that boundary. In other words, how the middleware architecture might be able to support adaptable applications with real-time and dependability requirements, that span entire cities, and different computational systems and paradigms. A middleware for the internet of things, in the true sense of the word.
Abbreviations

**ACK**  Acknowledgment.

**CPU**  Central Processing Unit.

**LQER**  Link Quality Estimation based Routing.

**PLER**  Probabilistic Latency Estimation based Routing.

**QoS**  Quality of Service.

**WSN**  Wireless Sensor Network.
Bibliography


