A RESTFUL BASED ARCHITECTURE FOR RECONFIGURABLE EXPERIMENTAL WIRELESS SENSOR NETWORK TESTBED

Adnan Bekan

Master Thesis
Jožef Stefan International Postgraduate School
Ljubljana, Slovenia

Supervisor: Assoc. Prof. Dr. Mihael Mohorčič, Jožef Stefan Institute, Jamova 39, SI-1000 Ljubljana, Slovenia

Evaluation Board:
Asst. Prof. Dr. Tomaž Javornik, Chairman, Jožef Stefan Institute, Jamova 39, SI-1000 Ljubljana, Slovenia
Asst. Prof. Aleš Švigelj, Member, Jožef Stefan Institute, Jamova 39, SI-1000 Ljubljana, Slovenia
Assoc. Prof. Dr. Mihael Mohorčič, Member, Jožef Stefan Institute, Jamova 39, SI-1000 Ljubljana, Slovenia

MEDNARODNA PODIPLOMSKA ŠOLA JOŽEFA STEFANA
JOŽEF STEFAN INTERNATIONAL POSTGRADUATE SCHOOL

Adnan Bekan

A RESTFUL BASED ARCHITECTURE FOR RECONFIGURABLE EXPERIMENTAL WIRELESS SENSOR NETWORK TESTBED

Master Thesis

ARHITEKTURA ZA NASTAVLJIVO EKSPERIMENTALNO BREZŽIČNO SENZORSKO OMREŽJE NA PODLAGI RESTFUL STORITEV

Magistrsko delo

Supervisor: Assoc. Prof. Dr. Mihael Mohorčič

Ljubljana, Slovenia, September 2015

This work is dedicated to my parents.

For their endless support and encouragement.

Acknowledgments

First, I would like to thank my supervisor, Assoc. Prof. Dr. Mihael Mohorčič, for his patience, support and guidance during my master studies. I could not have imagined having a better advisor and mentor for my master study. Then, I would like to thank my working supervisor Carolina Fortuna for guiding my research for the past several years and helping me to develop research background. I appreciate all her contributions of time, ideas and motivation to make my master degree.

Special thanks go to group of co-authors and collaborators, and my colleagues from the Jožef Stefan Institute, for many valuable discussions and contributions.

Next, I would like to thank the members of the Evaluation Board of my master thesis, Asst. Prof. Dr. Aleš Švigelj and Asst. Prof. Dr. Tomaž Javornik for their support and valuable comments.

Special thanks go to Melisa Junuzović for the support, patience and understanding during these years. She was always motivating me and encouraging through the good and bad times.

Finally, I would like to thank my parents for their support and encouragement during my studies.

Abstract

Typically, experimental WSN deployments can be found in research institutions and consist of few tens to few hundreds of low cost, low-power devices with limited processing and communication capabilities. These are typically centrally controlled and have a two or three-tier architecture. A relatively small number of these deployments are based on constrained devices and use a wireless management network, mostly because of the unreliable communication it enables. However, in some cases, such management networks are the only option due to the target location of such testbeds: outdoors, on light posts, buildings, etc.

In this thesis we propose a single-tier architecture for RESTful-based fully reconfigurable wireless sensor network testbed, which (i) can be used for experimentally driven research and development; (ii) should be deployable on an ad-hoc basis in any target environment; (iii) should be remotely configurable and controllable using simple RESTful APIs; (iv) has to be able to reconfigure and support easy experimentation and testing of standard protocol stacks (i.e. uIPv4 and uIPv6) as well as non-standardized clean-slate protocol stacks. In order to achieve this, we identified a list of challenges and requirements for designing a wireless sensor network testbed with optimal configuration to enable experimenting with new applications in, for instance, smart cities, industrial and building automation environments, environmental monitoring, etc.

As a practical part of the thesis we implemented and evaluated the performance of the proposed architecture on a sensor network consisting of VESNA platform nodes. The reference implementation of the architecture uses a dual-stack Contiki OS with the ProtoStack tool for dynamic composition of services. The parameters of the protocol stacks, standardized or non-standardized, can be remotely reconfigured through easy to use CoAP handlers. Additionally, we are able to fully reconfigure clean-slate protocol stacks at run-time. The architecture enables easy set-up of the network by using a protocol that automatically sets up a multi-hop network (i.e. RPL protocol) and it enables reconfiguration and experimentation by using a simple, RESTful interaction with each node individually.

For the validation of the reference implementation, we performed a set of uplink/downlink experiments on the management network and also executed localization experiment for the validation of the experimental network. The obtained experimental results of transmission and relocation times are in line with the expected values, proving that the proposed architecture provides a significantly faster way of deployment and configuration of WSN experimental testbeds compared to traditional deployment of sensor networks relying also on wired infrastructure.

Povzetek

V raziskovalnih organizacijah pogosto naletimo na eksperimentalna brezžična senzorska omrežja, ki jih običajno sestavlja od nekaj deset do nekaj sto nizkocenovnih naprav z majhno porabo in omejenimi računskimi in komunikacijskimi zmogljivostmi. Ta omrežja so navadno centralno krmiljena in imajo dvo- ali trokrožno arhitekturo. Razmeroma majhno število teh postavitev je zasnovanih na napravah z omejenimi zmogljivostmi in brezžičnem omrežju za upravljanje, v največji meri zaradi nezanesljivosti komunikacije. Seveda pa je v mnogih ciljnih obratovalnih okoljih brezžično omrežje za upravljanje edina možnost za izvedbo eksperimentalnega brezžičnega senzorskega omrežja, še posebno na prostem, na drogovih javne razsvetljave, na stavbah itd.

V magistrski nalogi smo predlagali enokrožno arhitekturo za povsem nastavljivo eksperimentalno brezžično senzorsko omrežje na podlagi RESTful storitev. To omrežje (1) je namenjeno eksperimentalno usmerjenim raziskavam in razvoju, (2) se lahko postavi na zahtevo v poljubnem ciljnem obratovalnem okolju, (3) mora podpirati nastavljanje in nadzor na daljavo preko enostavnih RESTful aplikacijskih vmesnikov ter (4) se mora biti sposobno preoblikovati in podpirati enostavno izvajanje eksperimentov in preizkušanja standardiziranih (npr. uIPv4 in uIPv6) kot tudi nestandardiziranih komunikacijskih protokolnih skladov. Identificirali smo vrsto izzivov in zahtev za načrtovanje brezžičnih senzorskih omrežij z optimalno konfiguracijo za eksperimentiranje z novimi aplikacijami za pametna mesta in stavbe, avtomatizacijo industrijskih okolij, nadzor okolja in drugo.

V praktičnem delu magistrske naloge smo realizirali in ovrednotili delovanje predlagane arhitekture v senzorskem omrežju, zasnovanem iz vozlišč na platformi VESNA. Referenčna izvedba arhitekture uporablja modificiran operacijski sistem Contiki z dvojnim protokolnim skladom in orodje ProtoStack za dinamično sestavljanje komunikacijskih storitev. Parametre standardiziranih ali nestandardiziranih protokolnih skladov lahko nastavljamo na daljavo z za uporabo enostavnimi upravljavci CoAP. Med samim izvajanjem lahko prenastavimo tudi celoten protokolni sklad. Arhitektura omogoča enostavno vzpostavitev omrežja z uporabo protokola, ki samodejno vzpostavlja omrežje z več skoki (tj. protocol RPL), ter nastavljanje in eksperimentiranje z uporabo preproste RESTful interakcije z vsakim posameznim vozliščem.

Za potrditev referenčne izvedbe smo izvedli niz poskusov na navzgornjih in navzdolnjih povezavah omrežja za upravljanje ter eksperiment lokalizacije na eksperimentalnem omrežju. Doseženi eksperimentalni rezultati za čas prenosa in premeščanja informacijskih blokov so v skladu s pričakovanimi vrednostmi in dokazujejo, da predlagana arhitektura v primerjavi z obstoječimi rešitvami, ki običajno vključujejo tudi žično infrastrukturo, omogoča bistveno hitrejše postavljanje in nastavljanje eksperimentalnih brezžičnih senzorskih omrežij.

Contents

List of Figures

List of Tables

Abbreviations

ARM	. . .	Advanced RISC Machines
ADC	. . .	Analog-to-Digital Converter
API	. . .	Application Programming Interface
BLE	. . .	Bluetooth Low Energy
CSMA	. . .	Carrier Sense Multiple Access
CPU	. . .	Central Processing Unit
CoAP	. . .	Constrained Application Protocol
DAC	. . .	Digital-to-Analog Converter
DHCP	. . .	Dynamic Host Configuration Protocol
ELF	. . .	Executable and Linkable Format
HTTP	. . .	Hyper Text Transfer Protocol
ISM	. . .	Industrial, Scientific and Medical
IrDA	. . .	Infrared Data Association
I2C	. . .	Inter-Integrated Circuit
ICMP	. . .	Internet Control Message Protocol
IETF	. . .	Internet Engineering Task
IoT	. . .	Internet of Things
IP	. . .	Internet Protocol
6LoWPAN	. . .	IPv6 over Low Power Wireless Personal Area Networks
JSON	. . .	JavaScript Object Notation
JSI	. . .	Jožef Stefan Institute
LQI	. . .	Link Quality Indicator
M2M	. . .	Machine to Machine
MTC	. . .	Machine Type Communications
MTU	. . .	Maximum Transmission Unit
MAC	. . .	Media Access Control
NDP	. . .	Neighbour Discovery Protocol
OTAP	. . .	Over the Air Programming
PRR	. . .	Packet Reception Ratio
PAN	. . .	Personal Area Network

PHY	...	Physical layer of the OSI model
RAM	...	Random Access Memory
ROM	...	Read Only Memory
RSSI	...	Received Signal Strength Indicator
RISC	...	Reduced Instruction Set Computing
RPC	...	Remote Procedure Call
REST	...	Representational State Transfer
RFC	...	Request for Comments
RPL	...	Routing Protocol for Low-Power and Lossy Networks
SD	...	Secure Digital
SNC	...	Sensor Node Core
SNE	...	Sensor Node Expansion
SNR	...	Sensor Node Radio
SPI	...	Serial Peripheral Interface Bus
SRD	...	Short Range Devices
SNMP	...	Simple Network Management Protocol
SoC	...	System on a Chip
TCP	...	Transmission Control Protocol
TVWS	...	TV White Spaces
UHF	...	Ultra-High Frequency
URI	...	Uniform Resource Identifier
UART	...	Universal Asynchronous Receiver/Transmitter
USB	...	Universal Serial Bus
USRP	...	Universal Software Radio Peripheral
UDP	...	User Datagram Protocol
VESNA	...	Versatile platform for Sensor Network Applications
VHF	...	Very High Frequency
WSN	...	Wireless Sensor Network

Chapter 1

Introduction

In this introductory chapter, we start with the motivation behind this thesis and the formulation of the problem addressed. Then, we define the goals of this thesis and provide a brief overview of related work. Finally, in the last subsection we introduce an outline of the thesis.

1.1 Motivation

Different aspects of wireless sensor networks (WSN), such as their single and multi-hop connectivity, energy consumption and system level aspects have been well studied for more than a decade now. More recently, there are signals of broader industrial adoption in real operating environments through the Internet of Things (IoT) concept and machine type communications (MTC) paradigm. We are witnessing that WSNs are starting to play an increasingly important role in monitoring and automatizing our houses and cities. One of the reasons why their adoption still seems to be slow is the fact that the majority of the past studies was only theoretical, with relatively few results verified on actual testbeds typically comprised of homogeneous and purposely designed sensor nodes. Even though projects such as Arduino and RaspberryPi significantly lowered the experimentation barrier, setting up and tuning reliable fully operating ad-hoc network of sensor nodes in a targeted operating environment, for instance in an industrial warehouse or a production plant, is still a challenging and time consuming task. These problems with the WSN deployments in production environment progressively attract research community attention, leading to an increasing number of experimental WSN testbeds.

Typical experimental WSN deployments consist of a few tens to a few hundreds of low cost, low-power devices with limited communication capabilities. These are typically centrally controlled and have a two or three-tier architecture [1], as it is depicted in Figure 1.1. The two-tier architecture comprises WSN devices (tier1) and the wired backbone to the server (tier2). WSN devices are organized in a flat architecture in this case. The three-tier architecture includes an additional gateway tier that enables a hierarchical WSN architecture. The three-tier architectures tend to be less energy efficient as the middle tier contains a significant number of high-power devices. An example of two-tier sensor network deployment is MIRAGE from Intel Berkeley and of three-tier are WISBED, TWIST and SmartSantander.

These deployments are controlled and managed through a wired management network and are mostly used for experimental research on wireless protocols. A number of more recent sensor deployments and testbeds such as SmartSantander [2], CitySense [3] and LOG-a-TEC [4] are located outdoors. To reduce the deployment costs, outdoor testbeds

typically use a wireless management network [1]. More recently, a trend towards mobile wireless sensor network testbeds such as MOTEL, CONET-IT and RoombaNet can be noticed [5]. Obviously, due to mobility, these testbeds are also controlled through a wireless management network.

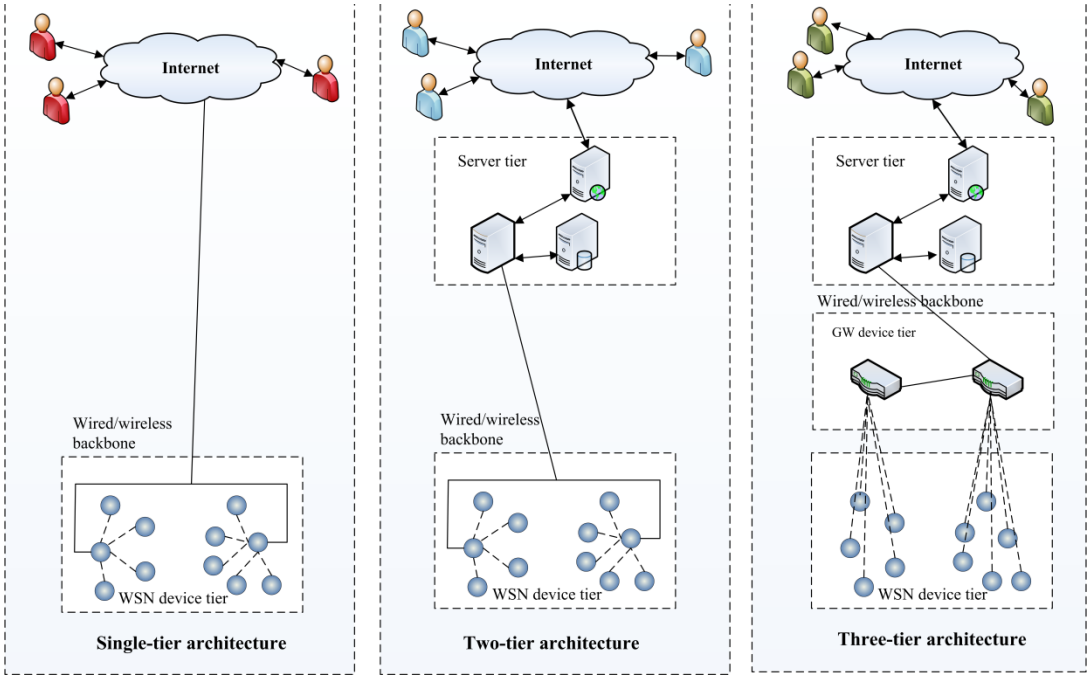

Figure 1.1: Typical WSN testbed architectures.

Independent of their architecture, type of management network or degree of mobility, the deployments dedicated to sensor data collection in some particular application domain such as IoT-Lab [6], WISEBED [7] and SmartSantander typically provide RESTful [8] interfaces through which the data is collected. The deployments dedicated to experimental research and development of new wireless technologies, e.g. NITOS, w-iLab.t and TWIST, provide non-RESTful interfaces such as direct ssh access to each node.

Existing two-tier and three-tier experimental testbeds require additional layer of devices for enabling full connectivity to the Internet and control interface to the end-user. To move such complex system architecture for the experimental purposes to the environments such as industrial warehouse or production plant can be complicated and time consuming. Even to reproduce such architectures is in most of the cases too expensive and not worth the effort for a short-term period.

1.2 Problem Formulation

The adoption of IoT/M2M in actual application areas such as smart cities, industry and building automation could be significantly faster if a cost-effective way of deploying reliable pilot wireless communication networks in actual operating environment was available. Current off-the-shelf solutions tend to be unreliable while custom built solutions by professionals require significant investment costs in infrastructure. The design of a new WSN with optimal configuration for industrial or building automation can thus be very challenging.

The purpose of the thesis is to define and evaluate an architecture for a RESTful-based fully reconfigurable experimental WSN testbed, suitable for fast on-site deployment, demonstration and testing in real operating environment that can foster the adoption of WSNs in production environments. Such testbed should be comprised of easily configured low-power devices that can be integrated into the working environment without interfering with the existing infrastructure.

The working hypothesis is that a system architecture which enables (i) an ad-hoc deployment of an experimental network of sensor nodes and (ii) custom configuration of the wireless solution in the actual target production environment rather than in a simulator or a lab will result in a significantly faster way of deployment of WSNs and in testing and validation of various solutions in real operating environments.

1.3 Goals of the Thesis

There is a significant body of work on large scale smart city, smart building or industrial adoption of sensor networks through the Internet of Things (IoT) concept and machine type communications (MTC) paradigm where WSNs are starting to play an increasingly important role in monitoring and automatizing our cities and houses. Such complex operating environments, however, call for careful and thorough testing and validation of new solutions early in the development process and in close to targeting operating environment essentially requiring experimental testbeds.

We aim to propose an architecture for a single-tier WSN testbed for experimentally driven research and development that can be (i) deployed on an ad-hoc basis in any target environment and (ii) remotely configured and controlled using simple RESTful APIs. The practicability of proposed architecture will be evaluated using a reference implementation consisting of VESNA platforms. Thus we will extend the predominant practice of theoretical and simulation-based performance evaluation of proposed solutions and enable broader adoption of experimentally driven research and development.

The main goals of the thesis are the following:

- Propose an architecture for RESTful-based fully reconfigurable wireless sensor network testbed, which (i) can be used for experimentally driven research and development; (ii) should be deployable on an ad-hoc basis in any target environment and (iii) should be remotely configurable and controllable using simple RESTful APIs; (iv) has to be able to reconfigure and support easy experimentation and testing of standard protocol stacks (i.e. uIPv4 and uIPv6) as well as non-standardized clean-slate protocol stacks.
- Develop software libraries that enable remote reconfiguration of the protocol stack parameters through the RESTful API and enable full reconfiguration of clean-slate protocol stacks in run-time.
- Develop software libraries for the RESTful-based full control of device's resources and the experiment.
- Provide reference implementation of proposed architecture using VESNA platforms.
- Evaluate the performance of a VESNA platform-based experimental testbed in terms of throughput and experiment setup.

1.4 Overview of the Thesis

This thesis is organized in six core chapters with the content of each chapter summarized as follows.

Chapter 1 presents motivation of the work, formal presentation of the problem and the goals of the master thesis.

Chapter 2 introduces background of wireless sensor networks, key enablers for building a network infrastructure and an overview of hardware and software solutions for WSN. Related work on some of already existing experimental WSN testbeds is also given within this chapter.

Chapter 3 presents challenges and requirements for an architecture for RESTful-based experimental testbed and finally presents the proposed solution for such testbed.

Chapter 4 presents in detail a reference implementation of the proposed system architecture starting with dual-stack functionality and programming models. Then, with sequence diagrams it depicts all enablers and procedures for monitoring, control and composition of WSN experimental testbed. Finally in the last section it presents the deployment and implementation of WSN experimental testbed.

Chapter 5 details the implementations and execution with results of three experiments in WSN testbed. First we perform transceiver selection experiment for the reliable and fast management network. The second experiment was performed for validation of speed and reliability of the wireless management network. The third experiment was performed with results for the validation of the experimental network.

Chapter 6 presents the conclusion of this work and identifies some of the possible future improvements.

Chapter 2

Background and Key Enablers

In this chapter we introduce the background on wireless sensors networks, internet protocol and REST architecture. After that we provide a brief overview of already existing embedded Operating Systems, well-known WSN hardware platforms and some of the already existing WSN experimental testbeds.

2.1 Wireless Sensor Networks

In a few years almost every electronic device will be accessible via Internet and our environment will be enriched with a large number of sensor nodes that can form Wireless Sensor Networks (WSNs). These nodes are typically low power devices equipped with a variety of sensors that are able to interact with the physical world. Furthermore, these devices are able to interconnect with each other and to communicate with the outside world over the Internet.

In [9] authors define the Wireless Sensor Network as a network of devices, denoted as nodes, which can sense the environment and communicate the information gathered from the monitored field (e.g., an area or volume) through wireless links. Some of these devices are deployed in inaccessible areas, therefore a node does not only have a sensing component but also storage, communication and processing capabilities. Nodes in a WSN can be mobile or stationary, homogeneous or heterogeneous, and their capabilities can vary widely. Moreover, each node in the network can communicate with its neighbours and the sink node (edge-router) that allows dissemination of data to other networks.

Due to the rapid development of sensor nodes, the currently available nodes are becoming more sophisticated in terms of CPU power, memory, energy efficiency and connectivity. Software, which runs on each node, can be reprogrammed using simplified application interfaces (API) for wired or wireless reprogramming.

When deploying a WSN network, among the standard requirements we typically need: network accessibility, content accessibility and network management [10].

- *Network accessibility*

In the WSN every device should be able to communicate to each other without human intervention (so-called machine-to-machine concept - M2M) and to interact with an end user. To achieve this, every sensor node that constitutes wireless network must support simple and unique way of addressing; even more, this type of network should enable that every user has direct access to each node in the network. In order to make this feasible, WSN must equate with existing computer networks so that every device which supports

the same type of addressing can be part of this network. By making each sensor node a directly addressable and accessible network entity, the system provides an easy way for discovering, controlling and invoking services from the sensor network. Users can interact with the system at high level of abstraction using standards-based communication technology that blends with the existing stack used by the Internet (HTTP/TCP/IP). In addition, nodes that build WSN network will be equipped with multiple communication interfaces such as Wi-Fi and Bluetooth low energy (BLE) interface, which will provide an even better network accessibility.

- *Network management*

Sensor networks should be easy to set up and use even for non-network experts and for users without advanced experience in programming. Users who would like to deploy sensor network should only need to provide Internet connectivity and source of energy while everything else should be automatically configured. That includes automatic address allocation and creating sensor network topology based on algorithms for ad hoc routing protocols. This self-configured network should be able to detect potential problems in network and try to find a solution for the same problems as specified or encountered in the past so that the end user will have fewer problems with the network structure. Still, advanced users would have options to make their own configuration for the entire network and they will be able to create their own static network topology.

- *Content accessibility*

Sensor networks are long running distributed computing systems that consist of a collection of sensor platforms working together to collect information about various phenomena and physical quantities, for instance, wind speed, wind direction, light, temperature, humidity and other relevant data according to specific application. In this network end users expect to have simple access to nodes and to have the ability to retrieve data and metadata from specific sensor node. Creating independent sensor network, where user can collect information accurately and reliably, enables building soft real-time service platforms such as data analytic, visualization and automation platforms. Employing these platforms every user can analyse incoming data, set triggers for real-time offset detection and create warning services.

2.2 IEEE 802.15.4

The IEEE 802.15.4 standard was created for low-power devices that operate in the 868 MHz, 915 MHz and 2.4 GHz frequency bands [11] and it specifies physical layer (PHY) and medium access control sublayer (MAC). This would mean that WSN networks using IEEE 802.15.4 can operate in three license-free industrial scientific medical (ISM) frequency bands. In the 2003 revision, the data rates supported by this standard were 20, 40 and 250 kbps respectively to the mentioned frequency bands. Few years later the 2006 revision improved the maximum data rates of the 868 and 915 MHz bands to support 100 and 250 kbps as well.

The address of a node in IEEE 802.15.4 wireless sensor network can be formed with 64bit extended or 16bit short address space. The PHY layer provides communication between radio transceiver and MAC layer. It provides two main services, PHY data and management service. The MAC layer provides a link in between the service specific sublayers and the PHY layer. Like the PHY layer, the MAC sublayer also provides two services, the MAC data service and the MAC management service [12].

Standard by itself offers two topology modes, peer-to-peer and star. In the star topology all the communication occurs trough the Personal Area Network (PAN) coordinator, while in peer-to-peer topology devices communicate directly with each other [13]. For the star topology it is specific that it supports two modes; the first one is synchronized and the second one is unsynchronized. In the synchronized mode a device can track beacons to synchronize with the coordinator. In particular, synchronization is good for polling, energy saving and slotted channel access. In unsynchronized mode, on the other hand, device can have direct access to the channel immediately when no activity has been detected. Data transfer between PAN coordinator and device in all topologies of IEEE 802.15.4 networks is always initiated by the device. Leaving control of data transmission to the device, energy savings can be maximised and thus extend battery life time.

2.3 IP in Wireless Sensor Networks

Internet protocol is a connectionless protocol where connection setup is not performed and it is not guaranteed that packets will arrive at the destination [14]. The Internet Protocol exists in two versions: IPv4 and IPv6. Primarily, IPv6 was introduced to solve the problem of the exhaustion of IPv4 addresses. Therefore, longer address space of IPv6 of 128 bit will provide 10^{38} addresses, instead of $4{\times}10^{9}$ possible addresses with the IPv4. Adopting IPv6, each device in network will be able to obtain a unique address and will be directly accessible avoiding many workarounds such as port mapping, address translation and gateways in the WSN.

Internet Protocol (IP) in particular is good for WSN network because it enables interoperability with existing networks that are based on the same protocol. With the IP, nodes in WSN network will be able to communicate with each other and the Internet without any additional hardware and translation software. As it is depicted in Figure 2.1, IP enables interconnection between many devices (e.g. laptops, smartphones, servers, etc.) in local networks and the Internet.

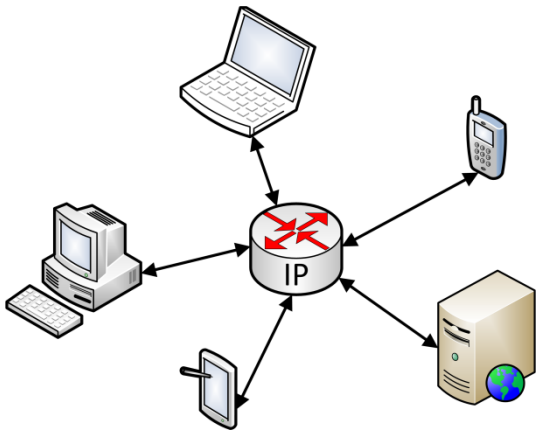

Figure 2.1: Interoperability of IP architecture.

Additional features of IP protocol:

- It is already standardized by IETF group
- It is supported by many existing Operating Systems (OS) including, Linux, OS X, Windows, Contiki OS, TinyOS etc.
- It supports dynamic addressing using Dynamic Host Configuration Protocol (DHCP).
- It enables internet-standard protocol for device management named Simple Network Management Protocol (SNMP)

2.3.1 IPv6

Internet Protocol Version 6 (IPv6) is the successor of IPv4 as the network protocol for the Internet, which addresses most limitations of IPv4. To overcome the decrease of unallocated address space and in anticipation that networked devices will vastly outnumber conventional computer hosts, IPv6 expands the IP address space from 32 to 128 bits [15]. To reduce the packet transmission overhead and improve the throughput, IPv6 routers do not perform packet fragmentation. Therefore, IPv6 increases the maximum transmission unit (MTU) requirement from 576 to 1280 bytes.

Security in IPv6 network is enabled with IPSec that provides confidentiality, authentication and data integrity. Other technical benefits of IPv6 protocols include; simplified and expanded use of multicast addressing and optimisation for delivery services. Two most important supporting protocols for IPv6 are Internet Control Message Protocol (ICMPv6) and Neighbour Discovery Protocol (NDP).

- Internet Control Message Protocol
ICMPv6 is a successor of ICMP and it is defined in RFC 4443. The main functionalities of this protocol are to report errors in processing packets, network diagnostic and other inter-layer functions. ICMPv6 messages are classified in two categories, error and information messages. They are transported by IPv6 packets with next header value set to 58 for their easier identification.

- Neighbour Discovery Protocol
NDP protocol is used with Internet Protocol version 6 and it operates in the second layer of OSI model. This protocol is defined in RFC 4861 and its main functionalities are discovering other nodes on the link, determine link layer addresses of other nodes, detect duplicated addresses, address prefix discovery etc. To successfully support the listed functionalities, NDP defines five ICMPv6 packet types that are: router solicitation, router advertisement, neighbour solicitation, neighbour advertisement, and network redirects.

2.3.2 6LoWPAN

By default IPv6 routers do not perform fragmentation thus demanding the support for MTU size of 1280 bytes. Adaption of IPv6 for low power devices that are based on low power radio links can thus be challenging. One of the standards defined for low power radio links is IEEE 802.15.4 standard and it only supports 127 bytes of maximum payload. To address proper solution for this problem 6LoWPAN group has defined an adaptation layer that sits between layers 2 and 3 just below the network layer where IPv6 is located. 6LoWPAN defines a header encoding to support fragmentation when

IPv6 datagrams do not fit within a single frame and additionally compresses IPv6 headers to reduce the header overhead [11].

6LoWPAN is a networking technology, and is used in embedded devices running on a microcontroller in the device. There are three different models typically used to embed such a wireless protocol solution: single-chip, two-chip and network processor solutions [13].

Single-chip solution is based on SoC radio technology where transceiver and microcontroller are integrated together with other components. This solution minimizes costs, size and complexity of embedded application.

In two-chip solution microcontroller and radio transceiver are separated. The protocol stack is integrated with application and it is running on a microcontroller. Communication with radio transceiver is enabled with serial peripheral interface (SPI). This solution is good where application complexity and performance requirements are high.

Finally network processor solution requires microcontroller for the embedded application and SoC radio chip for the protocol stack. Communication in between these units is enabled over SPI interface. This solution is good for project where minimum engineering effort is required.

2.4 Representational State Transfer (REST)

Representational State Transfer (REST) is a coordinated set of architectural constraints that attempts to minimize latency and network communication while at the same time maximizing the independence and scalability of component implementations [8, 16]. REST improves network efficiency by enabling cache mechanism and reuse of interactions, simplifies system architecture by using uniform interfaces in between components, and by adding a layered system style it improves the behaviour for Internet-scale requirements. In other words, REST is software architecture style for building scalable web services.

Typically, but not always, REST is implemented on Hyper Text Transfer Protocol (HTTP) and it is used instead of complex mechanisms such as Remote Procedure Call (RPC) or Simple Object Access Protocol (SOAP). To distinguish between resources, which are involved in an interaction between components, REST uses resources identifiers. All REST components perform actions on representation of a resource that consists of a sequence of bytes (content) and representation metadata to describe those bytes. For example, using Uniform Resource Identifiers (URI) in web-based applications any information that can be named is a potential resource and such resource can be called RESTful resource. RESTful is used to refer to web-services that are following the REST architecture.

Primary REST connectors are client and server. The essential difference between the two is that a client initiates communication by performing a request, whereas a server listens for connections and responds to requests. An important part of REST architecture is a well-defined interface to enable communication between components. In particular case of HTTP, a control interface is defined by a set of methods such as POST, GET, PUT and DELETE.

2.5 Constrained Application Protocol (CoAP)

The Constrained Application Protocol (CoAP) is an application-layer protocol designed to provide a REST-like interface, but with a lower cost in terms of bandwidth and implementation complexity than HTTP-based REST interfaces [17]. The protocol is designed for machine-to-machine (M2M) communication and devices that often have 8-bit microcontrollers and small amount of RAM and ROM.

CoAP provides a request/response interaction model between server and client, and supports built-in discovery of services and resources. CoAP is very similar to the HTTP protocol, because it adopts key concepts of the Web such as URIs resources with REST architecture and Internet media types. The main difference in between the two protocols is that CoAP uses a compact binary representation designed to be easily parsed and UDP transportation protocol instead of TCP.

To enable communication between server and client, server makes resources available under a URL and clients access these resources using methods such as GET, PUT, POST, and DELETE. Particularly interesting feature, which cannot be found in HTTP, is publish/subscribe mechanism called "observing resources" [18]. Observing resources enables push notification messages to the subscribers, whenever the observed resource changes. Since HTTP and CoAP share the REST model, they can easily be connected using application-agnostic cross-protocol proxies. In such case end user may not even notice that he accessed a sensor resource.

2.6 Operating Systems for WSN platforms

Unlike traditional operating systems such as Windows, OS X or Linux, the operating system for WSN platforms are optimized and tailored to the limited node hardware. These WSN operating systems multiplex hardware resources and provide a hardware abstraction to make developing of sensor applications simpler and more portable. In the WSN research community some of the most widespread operating systems, which are presented in more detail in the following, are TinyOS, Contiki, Mantis, and Nano-RK. Likewise, many other operating systems for WSN platforms exist.

A. **TinyOS** [19] is an open source, component based, and application-specific operating system targeting wireless sensor networks. TinyOS is an embedded operating system written in nesC programming language and it was developed in collaboration between University Berkley, Intel Research and Crossbow Technology. Footprint of the base the TinyOS can fit in less than 400 bytes of memory. TinyOS enables concurrent programs execution using full multitask solution with memory protection mechanism. Component library includes many features such as network protocols, distributed services, sensor drivers, and data acquisition tools. Latest version of TinyOS was upgraded with DIP (Dissemination Information Package), a new dissemination protocol for sensor networks. DIP is a data discovery and dissemination protocol that scales to hundreds of values. From the version 2.1.1 TinyOS also provides support for IPv6 protocol using 6LoWPAN optimization networking.

B. **Contiki** [20] is a lightweight open source OS for networked, memory-constrained system with particular focus on low-power WSN platforms. Contiki OS is written using C programing language and it was developed by Adam Dunkels in 2002. Contiki is a highly portable OS and it is built around an event-driven kernel and

provides preemptive multitasking that can be used at the individual process level. A typical Contiki footprint can fit in 40 kilobytes of ROM, which requires less than 2 kilobytes of RAM. Building blocks of OS include features like: multitasking kernel, preemptive multithreading, proto-threads, TCP/IP networking, IPv6, a Graphical User Interface, a web browser, a personal web server, a simple telnet client, and virtual network computing. To enable interconnection with already existing IP networks Contiki employs uIP solution. uIP is written in C and it supports TCP, UDP, ICMP, and IPv4/v6 protocols. Likewise, Contiki provides another lightweight layered protocol stack, called Rime [21], which is used for network-based communication and supports both best effort and reliable transmission. Applications are allowed to implement protocols that are not present in the Rime stack.

C. **MANTIS [22]** stands for MultimodAl system for NeTworks of In-situ wireless Sensors. It provides a new multithreaded cross-platform embedded operating system for wireless sensor networks. MANTIS is a lightweight and energy efficient operating system that is written in C programming language. Base footprint requires 14kB of ROM and less than 500 bytes of RAM. Key features of full OS image are energy-efficient scheduler for duty-cycling, a scheduler for preemptive multi-thread, and a network stack. Moreover, it supports diverse WSN platforms and remote management of sensor nodes through dynamic programming. Supporting additional hardware components on WSN platforms require component specific drivers implementation. In MANTIS each driver should implement four functions dev_read, dev_write, dev_mode and devo_ioctl. Afterwards, component can be used and controlled in the user application.

D. **Nano-RK** [23] is a reservation-based multitasking real-time OS for WSNs developed by Carnegie Mellon University in 2005. Nano-RK currently runs on the FireFly Sensor Networking Platform as well as the MICAz “motes” and is written using C programming language. Nano-RK includes a light-weight kernel (RK) with rich set of features, such as multitasking, multi-hop networking, priority-based scheduling, timeliness, extended WSN lifetime, application resource usage limits, and small footprint. Typical footprint of Nano-RK requires 2 Kb of RAM and 18 Kb of ROM. To ensure that task deadline is met, Nano-RK supports fixed-priority multitasking. Because dynamic allocation in Nano-RK is disallowed, before the application deployment developer must set task and reservation priorities. Nano-RK enables hard and soft real-time applications by the means of different real-time scheduling algorithms and for real-time data streaming over the network it supports socket-like abstraction.

2.7 WSN Hardware Platforms

A typical WSN hardware platform consists of sensors and actuators to enable interaction with the physical world; a communication interface to simplify the interaction; a processing unit that interacts with the peripheral components, usually a low-power microcontroller with a memory size of few kB; and a power source which can be a battery or other alternatives such as solar cells. Today, various WSN hardware platforms exist and new ones are being developed that are more sophisticated in terms of processing power, energy consumption and communication interfaces. An overview of few recently used platforms in experimental testbeds is given in this selection.

A. **The VESNA platform** [10] (developed by SensorLab) has been primarily designed for sensor network applications, although its modularity and expandability make it useful for other purposes, too. Due to its modular design and the availability of expansion connectors, it can be easily adapted to many applications. The main module of the node hosting the microcontroller is called the Sensor Node Core (SNC) module. The VESNA sensor platform is depicted in Figure 2.2. VESNA employs a STM32F103Rx microcontroller, having a 32-bit ARM Cortex-M3 processor core. The microcontroller has 64 kB of RAM and 512 kB of flash memory, and it supports a variety of interfaces: USB, RS232, UART, IrDA, SPI, I2C, SD/MMC, 12 bit ADC, DAC. The core module of the VESNA platform has two expansion connectors: one designed for Sensor Node Radio (SNR) modules, and one for Sensor Node Expansion (SNE) designed for various purposes. As radio expansion modules, sub-gigahertz and 2.4 GHz-band transceivers are available (e.g. using Texas Instruments CC1101/CC2500 and Atmel AT86RF212/RF231 radio transceivers), but alternatively other modules can be used as add-on boards connected to the radio connector. Examples of various expansion boards are the debugging and programming board, Ethernet to serial converter, WiFi to serial converter, proto-board modules and the additional power supply module.

Figure 2.2: VESNA sensor platform.

B. **The TelosB** [24]**,** sometimes also referred to as the TmoteSky, is developed and published to the research community by UC Berkley. It consists of MSP430 microcontroller and CC2420 radio transceiver. Microcontroller operates at 8MHz and has a 10kB of RAM and a 48kB of flash. For the data logging it supports 1MB external flash, where all the measurements can be stored. With an integrated on-board antenna, single-chip CC2420 radio transceiver that is IEEE 802.15.4 and ZigBee compliant provides up to 125 m of cover range. Power lifetime on the batteries is around 3 to 6 months depending on how often the radio transceiver is used for network communication. Its components cannot be enhanced, restricting use from its better radio transceiver/antenna to reach a longer distance. TelosB/TmoteSky is ultra-low-power wireless module designed to enable cutting-edge experimentation for the research community.

Figure 2.3: TelosB mote open-source platform.

C. **The iSense** [25] modular hardware and software platform for wireless networks is intended for both industry and research applications. In order to fit a wide variety of application demands the iSense hardware platform is made up of a number of modules that can be combined in various ways. Like this, functionality can be easily rearranged, and new features can be added by appending new modules. The hardware is arranged around the iSense CoreModule with an IEEE 802.15.4 compliant radio, a 32-bit RISC controller running at 16MHz, 128kB of memory and 512 kB of Flash, a highly accurate clock and a switchable power regulator. Moreover, CoreModule can be combined with a number of sensor modules (e.g. a thermometer, a light sensor, an accelerometer, a passive infrared sensor, a magnetometer and camera), different power sources including a solar power solution, a gateway and special purpose modules. With this platform application specific sensor nodes can be constructed just by putting together the required hardware modules. The hardware is supplemented with a modular operating and networking firmware that is based on object oriented programming.

Figure 2.4: iSense modular platform.

D. **The MPR2400 MICAz** [26] WSN platform was designed specifically for embedded sensor networks. MICAz is based on the low-power microcontroller Atmel ATmega128L, which runs MoteWorks from its internal flash memory. MoteWorks enables the development of custom sensor applications and it is optimized for low-power battery operated sensor devices. The MICAz platform comes with 802.15.4 compliant radio transceiver that operates at 2.4 GHz. Likewise, processor board (MPR2400) can be configured to simultaneously run sensor application and the network communications stack. The 51-pin expansion connector supports Analog Inputs, Digital I/O, I2C, SPI and UART interfaces. These interfaces make it easy to connect to a wide variety of sensor and data acquisition boards. Also, custom sensor and data acquisition boards are offered with this platform. Most common usage of MICAz platform is in indoor building monitoring, where high speed sensor data is required and in a large scale sensor networks.

Figure 2.5: MICAz mote.

2.8 WSN Experimental Testbeds

Recently, an increasing emphasis is being put on experimentally validated research, and a fair number of sensor-based testbeds exist such as Motelab [27], w-iLab.t [28], TWIST [29] and WISEBED [7], just to name a few. According to a recent survey [1], existing sensor-based testbeds are located mostly indoor and nodes are controlled through a wired management network such as USB or Ethernet. A number of sensor-based testbeds located outdoor such as SmartSantander [2], CitySense [30] and LOG-a-TEC [4] also exist.

Generally we distinguish two generic types of testbeds consisting of constrained devices such as sensor nodes. The first type is focusing on monitoring of some phenomena such as energy consumption, humidity, temperature, motion, sound, gases, etc. These kinds of testbeds are typically used by researchers to automatically acquire some data about the phenomena under study. Examples of such testbeds are SmartSantander and CitySense. The second type is aimed at experimentation and refers to sensor-based testbeds that support the development of new communication and networking technology by enabling experimentation with new algorithms and protocols. Motelab, TWIST and

LOG-a-TEC are examples of such testbeds. The following paragraphs survey a few of already existing monitoring and experimental WSN testbeds in more detail.

A. **MoteLab** [27] is a sensor network testbed platform developed at Harvard University. MoteLab consists of a 30 Ethernet-connected MICAz "motes" that are distributed across 3 floors inside the laboratory building. These sensor nodes are connected to the central server, which enables reprogramming, logging and web-interface for interaction with the nodes. Employing the web technologies it enables management of experiments and direct access to the nodes, even during the experiment. Besides, it is one of the first experimental testbeds that simplifies and accelerates application deployment and development by consolidating access to a large number of experimental devices. MoteLab web interface allows both local and remote users access to the testbed, and its scheduling and quota system ensure fair sharing of the resources.

B. **TWIST** [29] is a wireless sensor network testbed developed at the Technische Universität Berlin. TWIST is one of the largest academic testbeds. It consists of 204 motes, 102 of them are TelosB motes and the other 102 are eyesIFX. Motes are distributed inside the building across 3 floors, with the distance between them of 3m, forming a grid pattern. Over the USB hubs sensor motes are connected to several super nodes that ensure Ethernet backbone to the central server and control station. TWIST provides experiments with heterogeneous node platforms, support for flat and hierarchical setups and power supply control of the nodes. To enable experimentation control several Python scripts runs locally on the super nodes and provide functionalities like sensor node programming, executing power control, collecting debug and application data, and more. Invocation of these scripts is done by the control station using ssh network protocol.

C. **w-iLab.t** [28] is a generic wireless sensor testbed developed at the iMinds research institute. w-iLab.t is deployed at two different locations; 200 sensor nodes are distributed across iMinds premises in Ghent and additional 60 in w-iLab.t Zwijnaarde 5 km away from the first location. At the first location each node out of 200 is comprised of an embedded PC and Tmote Sky motes, while at the second location instead of Tmote Sky they employed RM090 wireless sensor module. w-iLab.t enables experiments with 802.11 a/b/g, 802.15.4 and 802.15.1 interfaces, moreover it allows flexible evaluation of hardware and software. To enable control and scheduling of experiments w-iLab.t uses web server and for root access to the devices it uses ssh network protocol.

D. **CitySense** [30] is an urban wireless sensor network testbed developed by researchers at Harvard University and BBN Technologies. The testbed consists of 100 embedded PCs with 802.11 a/b/g interfaces and various sensors used for weather and air-pollution monitoring. Nodes are deployed across the city on the light poles and buildings. For the management network they are using wireless network-based on Ubiquiti SR9 802.11 b/g mini modules. CitySense enables experiments with the various mesh routing protocols and urban environmental monitoring. Testbed management and experiment control is done by executing commands using ssh network protocol while on the other hand, collected sensor data can be accessed using the web server.

E. **LOG-a-TEC** [31] is an experimental wireless sensor network testbed developed by the SensorLab group at the Jožef Stefan institute (JSI). The testbed consists of 70-80 low power VESNA sensor nodes distributed at two different locations in the town Logatec and at the JSI campus. In both locations VESNA nodes are used, mostly deployed outdoors on the light poles and building facades. The testbed employs 802.15.4 interface for the wireless management network that enables remote monitoring, reconfiguration and experimentation. LOG-a-TEC enables experiments with various sensors for environmental monitoring, smarts grids and cognitive radio experimentation. LOG-a-TEC provides two ways of testbed management. First, is using non-RESTful web interface that enables monitoring, controlling and reprogramming of WSN devices and the second is using Python library that enables running of more sophisticated experiments such as spectrum sensing or power-allocation experiments.

Chapter 3

Architecture Design of WSN Testbed

In this chapter we first identify challenges and requirements for setting up an experimental testbed consisting of constrained devices. Then we propose architecture for a single-tier testbed for experimental research and development that can be (i) deployed on an ad-hoc basis in any target environment and (ii) configured and controlled using simple RESTful APIs.

3.1 Challenges

Current off-the shelf WSN solutions tend to be unreliable and unfit for fast and scalable deployments for experimental purposes, while custom built solutions by professionals require significant time, effort and investment costs in infrastructure. In this respect, the design of a new wireless sensor network with optimal configuration to enable experimenting with new applications in for instance smart cities, industrial and building automation environments has to consider a set of challenges outlined in the following subsections [32].

3.1.1 Resource-aware Experimentation

The first challenge regards the WSN devices which are typically small in size and cheap. This leads to a series of limitations such as limited memory, lower processing power and battery-based powering. Given these constraints, the first main challenge is designing an architecture that enables experimentation with as low overhead as possible. This translates into being able to control, debug and reconfigure the network under test using as few bits sent over the air as possible since it is known that the wireless interface consumes most of the energy of such devices [33]. Additionally, the time required for configuring an experiment should be reasonable. Therefore, besides minimizing the bits transferred over the air, the architecture should also reduce complex operations such as large and slow memory relocations on the embedded device itself whenever possible.

3.1.2 Context-aware Deployment and Configuration

The second challenge regards the target deployment of the experimental ad-hoc network. In real life operating environments indoor and outdoor alike, there are two main factors that affect the wireless connectivity, (i) the pre-existing wireless infrastructure as a potential source of interference and (ii) varying set-ups consisting of moving obstacles (e.g. people, furniture, merchandise) that may lead to significant variations in multi-path propagation. Assuming these constraints are given before setting up the testbed, it must

be possible to determine, for instance via spectrum sensing, the occupied bands and channels as well as the feasible location for deploying the sensors and the distance between them. Then, the deployed network has to be configured accordingly not to interfere with the existing wireless infrastructure and ensure that the chosen frequency band allows operational connections.

3.1.3 Remote Control and Optimization

After deploying the testbed and appropriately configuring it on the spot, it should be possible to assess the functioning and reliability of the resulting. A cost efficient and simplest way of realizing this is to support remote control and optimization through an easy to use API. One example of such implementation is to allow remote control over the web and exposing an easy to use RESTful API. This lowers the skill level required to write scripts for visualization, monitoring and configuring the testbed remotely compared to the traditional command line approaches.

3.2 Requirements for Remote Control and Optimization

Following the challenges identified in Section 3.1, we identified a set of common requirements and design goals for increasingly capable remote handling of the testbed architecture. By addressing these requirements, the resulting testbed architecture should lower the efforts in development and experimental evaluation of custom-built wireless solutions in challenging real-life environments and thus increase the innovation potential in wireless sensor networks.

3.2.1 Remote Monitoring and Diagnosis

The first important requirement from the architecture point of view for an ad-hoc testbed is to enable the remote monitoring of the radio and network parameters as well as diagnosis of the overall network. The nodes in remotely installed testbeds have to be able to provide on request information about their status, health and current configuration. For instance, ping times and round-trip times might be requested each few minutes from the remote monitoring application that assesses the overall performance of the tuned network. Based on these statistics, the network performance can be improved manually or automatically. Micro controller and surrounding temperature might be an important parameter to monitor, especially in industrial production environments, as these affect the performance and the lifetime of the nodes. An overall network configuration such as routing or neighbourhood tables as well as current protocol configurations should be provided on demand for performing remote diagnosis.

3.2.2 Remote Parameter Tuning

The second requirement from the architecture perspective for an ad-hoc testbed is to enable the remote tuning of parameters. These parameters can be grouped into (i) transceiver parameters such as transmit power, operating frequency and operating bandwidth, and (ii) protocol parameters such as contention window size for CSMA, time to live value for IP or the ceiling of the exponential back-off in TCP. These are all important for the remote configuration and tuning of the custom wireless solution. For instance, if the connection between two deployed nodes proves to be systematically unreliable, perhaps the transmit power has to be increased or the transmit channel

changed. After applying the changes, the behaviour of new configuration can be re-evaluated.

Remote parameter tuning can be performed at run-time or by rebooting the corresponding node so that the changes take effect. For the sake of experimentation speed, run-time reconfiguration should be supported for as many parameters as possible. This is also convenient for advanced algorithm development where dynamic interference mitigation or power allocation settings are chosen automatically because the system can perform timely and agile changes. For instance, with advanced power control algorithms such as the one proposed in [34], a run-time reconfigurable system can adapt the transmit power and thus maintain good links in dynamic environments where transmitters and obstacles appear and disappear.

3.2.3 Over the Air Software Updates and Upgrades

Next requirement from the architecture point of view for an ad-hoc testbed is to enable efficient over the air software updates and upgrades. Such updates are useful for debugging, upgrading existing functionality such as a protocol setup or adding new functionality [35]. In [10], the authors identify three types of required updates by experimental testbeds: OS/firmware upgrades, driver updates and application updates. OS/firmware upgrades are expected to occur when new versions of software are released or when a major flaw is discovered and needs immediate fixing. However, for experimental setups that require software modification, the need for over the air programming (OTAP) might be more frequent. In many cases, these upgrades can be achieved using dynamic linking, thus avoiding the need for realizing a full OS/firmware upgrade. In such cases, the file sent to the nodes of the testbed is relatively small compared to the full OS/firmware image. Performing a full OS/firmware upload for each application tends to be uneconomical.

3.2.4 Modular Stack Reconfiguration

The fourth requirement for the architecture of an ad-hoc testbed is to enable modular stack reconfiguration. While the first three requirements are mandatory for such systems, this last requirement is only needed for advanced users that, besides tuning and configuring protocol parameters, intend to reconfigure the protocols that form a stack as part of the design and development of their custom wireless solution. This can also be done via remote reprogramming in case the protocol stack has a monolithic implementation, however it is less energy efficient and more time consuming due to the higher number of bits sent over the air.

For modular implementations of protocol stacks such as the standards based IPv4/6 implementations in PicoMESH[1] or the clean-slate CRime [36], the entire protocol stack can be reconfigured on the node provided all the modules have been pre-installed. For instance, one can easily switch between IPv4 and IPv6 by just changing the layer three protocol implementation module while the MAC and transport layer protocols remain the same. This requirement supports the development of efficient communication protocols and algorithms that are generic and are independent of the operating system. Additionally, the experimental evaluation of cross layer [37] and cognitive networking [36] techniques can be made much easier with such functionality.

[1] PicoMESH repository - https://github.com/tass-belgium/picotcp-tinyOS

3.2.5 Mapping of Challenges and Requirements

Table 3-1 presents mapping between the challenges identified in Section 3.1 and the architecture requirements as identified in Section 3.2. It can be seen that by supporting remote monitoring and configuration as well as over the air reprogramming, more resource aware experimentation is facilitated, particularly because of the convenience of the final solution. Additionally, with careful design and optimal engineering, the number of bits sent over the air for experimentation can be reduced. Remote control and optimization can be achieved by enabling remote parameter tuning and software updates/upgrades.

Table 3.1: Requirements and challenges.

Requirements / **Challenges**	**Remote monitoring and diagnosis**	**Remote parameter tuning**	**Over the air software updates and upgrades**	**Modular stack reconfiguration**
Resource-aware experimentation	X	X		X
Context-aware deployment and configuration	X	X		X
Remote control and optimization	X	X	X	

3.3 Proposed System Architecture

In this section, we propose an architecture that enables a fully reconfigurable wireless sensor network testbed. The proposed architecture based on the requirements from Table 3.1 is able to reconfigure and support easy experimentation and testing of standard protocol stacks (i.e. uIPv4 and uIPv6) as well as non-standardized clean-slate protocol stacks (e.g. configured using Rime). The parameters of the protocol stacks can be remotely reconfigured through the RESTful API. Additionally, we are able to fully reconfigure clean slate protocol stacks at run-time. The architecture enables easy set-up of the network by using a protocol that automatically sets up a multi-hop network (i.e. RPL protocol) and auto-configure the management network, thus requiring no manual work [32]. This makes each experimental device a directly and uniquely addressable and accessible network entity. It enables reconfiguration and experimentation by using RESTful interactions with each individual node.

The resulting system topology is presented in Figure 3.1 and is comprised of two parts. On the right hand side we have the resulting experimental network and on the left hand side we have end-users that can easily over the Internet interact with the experimental WSN device using the REST architecture.

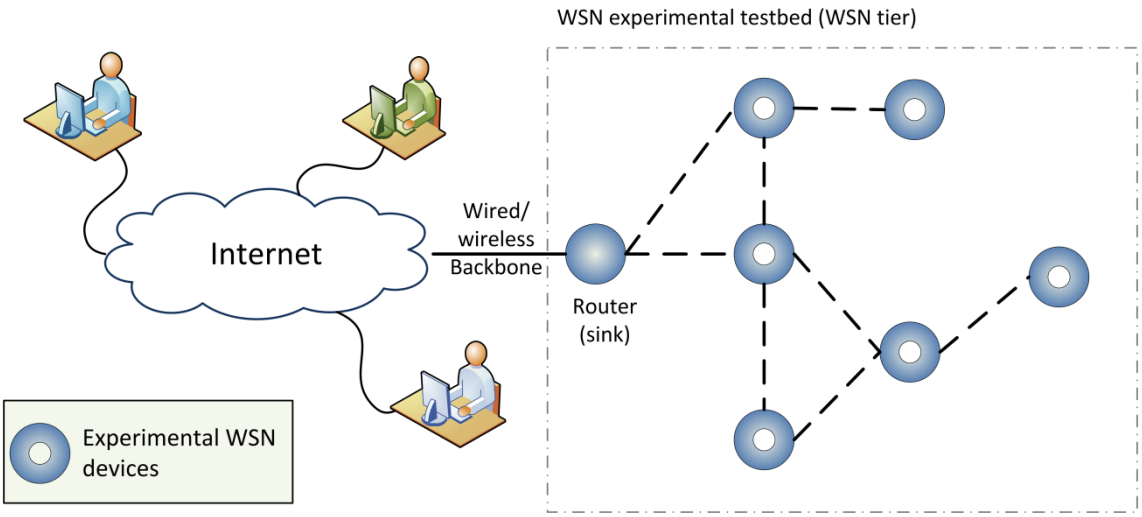

Figure 3.1: System architecture.

The experimental network consists of experimental devices and a router/sink block. Experimental devices in proposed architecture to avoid interference and simplify device management, should support two network interfaces one for management and the second for experimental network. While the experimental devices are only meant for performing experimentation, the router/sink has two functionalities. The first one is to seamlessly integrate experimental devices into the existing network, thus providing connection to the end-users via wired or wireless backbone link. The second functionality is to enable clustering of devices. Regarding the number of routers supported by the architecture, experimental devices can be organized with one router in flat or with many routers in hierarchical architecture.

This architecture provides an approach to describe, discover and invoke services from heterogeneous sensor networks. The two major advantages of such architecture are (i) that wireless sensor nodes can be deployed in an ad-hoc manner and (ii) that it enables custom configuration of the wireless solution in the actual real-life environment rather than in a simulator or a lab. This implies that with the proposed architecture it becomes much easier than before to develop and test a WSN-based application.

3.3.1 WSN Node Architecture

Figure 3.2 depicts the block scheme of an experimental WSN node. The node has to support (i) one or more configurable wireless transceivers for experimentation and/or management, (ii) a configurable protocol stack and/or a modular and configurable protocol stack and (iii) a monitoring, control and composition block. Additionally, there must be support for software upgrades to address the requirements in Section 3.2.3. The support for software upgrades can be realized in several ways, for instance by using custom firmware or operating system support. This aspect has been investigated in several recent works [35] which discuss in detail the possible designs and their trade-offs.

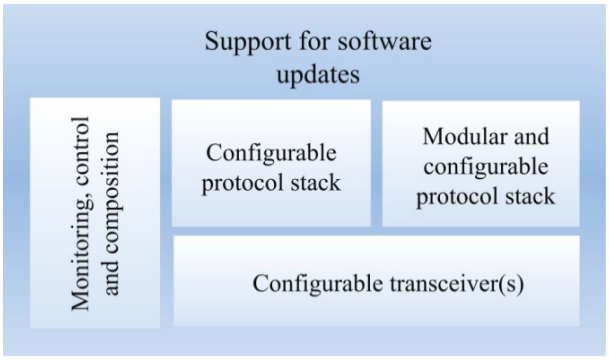

Figure 3.2: Block scheme of an experimental WSN node.

3.3.2 Reprograming and Reconfiguration Models

The basic reprogramming model involves the use of a bootloader to replace the whole image on the node (i.e. flash the node)[38], [39] and is the most suitable for OS/firmware upgrades. With multiple images it is possible to support multiple applications or implement failure recovery. This approach is used by all the wired testbeds to upload new experiments. The main drawback is large transfer size which is less suitable for wireless management networks. One way to reduce the transfer size is to use image compression [40].

More granular upgrades that may sometimes be more suitable for wireless management networks can be achieved by implementing functionality as applications, ran by the core OS. Virtual machine interpreters such as Mate [41] or Java based VM [42] have high overhead due to interpreted execution. This can be avoided by deploying applications as modules in native code. Pre-linked modules have near zero size overhead compared to monolithic application. Dynamically linked modules can be deployed to nodes with different OS images [43], [42]. However the overhead of additional metadata can be relatively large compared to the application code/data size, so the users should check if the overhead is acceptable.

As discussed in Section 3.2.3, for some application updates it is not needed to update the application program code. Component-based development requires splitting a monolithic OS image with application(s) into multiple components, which communicate via well-defined interfaces [44]. For instance, RemoWare permits, beside reconfiguration, also the addition of new components via dynamic linking [44].

3.3.3 Configurable Radio Transceivers

The configurable wireless transceiver has to support parameter reconfiguration (i.e. tx power, etc.) as discussed in Section 3.2.2. This means it has to be carefully selected to be closer to a white-box transceiver that can be reconfigured and can support a large variety of protocol stacks rather than to a black-box one where the transceiver and the protocol stack are tightly integrated and prevent configurations. In some cases, having a single transceiver may be sufficient. This implies that the management network (i.e. the network that performs monitoring, parameter tuning, etc.) is the same as the experimental network (i.e. the network to be optimized and tuned for the custom deployment). In this situation we are dealing with in-band monitoring and configuration.

To avoid in-band management that may interfere with the experiment, the architecture also supports two separate transceivers. In this situation, the management

and experimental networks can be clearly separated and configured to operate on non-overlapping channels or even different frequency bands, thus isolating the effects of the management-related communication overhead from the target production wireless network that is subject to customization. This is a preferable solution in most cases, especially when the experimental stack is not fully stable and needs remote debugging. If a single transceiver was used, a reprogramming error can also compromise the management network, and the remote access to the experimental network can be hindered.

Cheap and low-power transceivers tend to have limited support for frequency range. For instance, only operation in SRD 868 MHz or ISM 2.4 GHz bands may be supported, but not both at the same time by such a transceiver. In order to support both frequency bands and also additional ones such as TVWS (UHF, VHF-High, VHF-Low), more transceivers have to be added to the sensor node. This offers more flexibility in developing the solution but poses some additional hardware and software integration challenges. The alternative is to use more powerful software defined radios, however, for the time being, these are more expensive and not designed for operation with constrained devices (even though Embedded USRP is slowly changing this). As a generic architecture, we also have to consider the support of several interfaces to enable flexibility in experimentation.

3.3.4 Configurable Protocol Stack and/or Modular and Configurable Protocol Stack

An open (i.e. white-box) and configurable protocol stack running on top a configurable transceiver is the minimal architectural support for addressing the requirements from Section 3.2.2. For efficient developing and evaluation of variations of the same protocol or enabling advanced experimentation with cross-layer and cognitive networking techniques as discussed in Section 3.2.4, a modular and configurable protocol stack is desirable. These stacks should be able to run with minimum custom firmware as well as to integrate with existing operating systems for embedded devices. The firmware and/or operating system should enable software upgrades as discussed in Section 3.2.3, thus adopting existing solutions such as discussed in [35]. In some cases, using remote reconfiguration, the entire experiment can be remotely reconfigured. For instance, using run-time reconfiguration, the entire protocol stack can be reconfigured without flashing the node.

3.3.5 Monitoring, Composition and Control Block

The monitoring, control and composition block must be able to configure the transceivers and protocol stacks as described in Section 3.2. To enable the experimentation and control parts of the testbed, it must provide an easy way of remote monitoring, configuring and composing network resources. This is typically achieved with a simple API that follows the RESTful architecture. Monitoring can be simplified using the simple RESTful solution, end-user of the testbed can change the sampling rate of the sensor or can change the size of the buffer that stores the measurements in a very few steps. Also experimental parameters such as the frequency at which packets are sent, the number of retransmissions, transmitting power, receive channel filter bandwidth and carrier sense indicator, could be easily controlled.

Chapter 4

Implementation of RESTful-Based Experimental Testbed

In this chapter we discuss the design choices for implementation of a single-tier WSN experimental testbed, reprogramming and reconfiguration models used, and CoAP handlers for the management of RESTful-based experimental testbed.

4.1 Testbed Deployment

When building a new testbed from scratch, one is faced with selecting a desired hardware platform and a supported operating system. Heterogeneous testbeds may consist of several different hardware platforms. The challenges and requirements discussed in Section 3 should be taken into account when selecting the hardware and OS. For instance, with some operating systems it will be impossible to support dynamic linking and/or run time reconfiguration. When upgrading an already existing testbed, the guidelines presented in the previous sections still hold, however, there may be already existing constraints on the choice of hardware and software.

4.1.1 LOG-a-TEC 2.0 RESTful Testbed

The proposed reference implementation builds on our previous experience with outdoor testbed deployment and experimentation with spectrum sensing and cognitive radio in the LOG-a-TEC testbed [4]. As a result, we use the VESNA sensor platform which is a low power modular platform with several options for transceiver selection and the Contiki OS which is a commonly used sensor operating system that also enables modular updates rather than just monolithic updates. The VESNA core board contains a 32-bit ARM Cortex-M3 microcontroller running at up to 72 MHz CPU clock. It has 64 kB of RAM and 512kB of flash memory. Additional non-volatile memory is provided by a micro SD card.

As it is depicted in Figure 4.1, 20 VESNA sensor nodes are mounted at the campus of the Jožef Stefan Institute in Ljubljana, Slovenia. They cover approximately 3000 m2 of in-door and out-door space. This testbed is currently used for experiments with packet-based transmissions and dynamic network stack composition in wireless sensor networks. Each node in the testbed contains two radio transceivers, one bare bone for experimental network and one IEEE 802.15.4 compliant for the management network. Nodes at JSI campus are running the Contiki operating system version 2.7 with a dual, composable networking stack. A 6LoWPAN network using the IEEE 802.15.4 radio is used to control

and reprogram the nodes. Both networks can be operated simultaneously without interfering with each other as they operate at different frequencies. Each node is directly accessible from the Internet using IPv6.

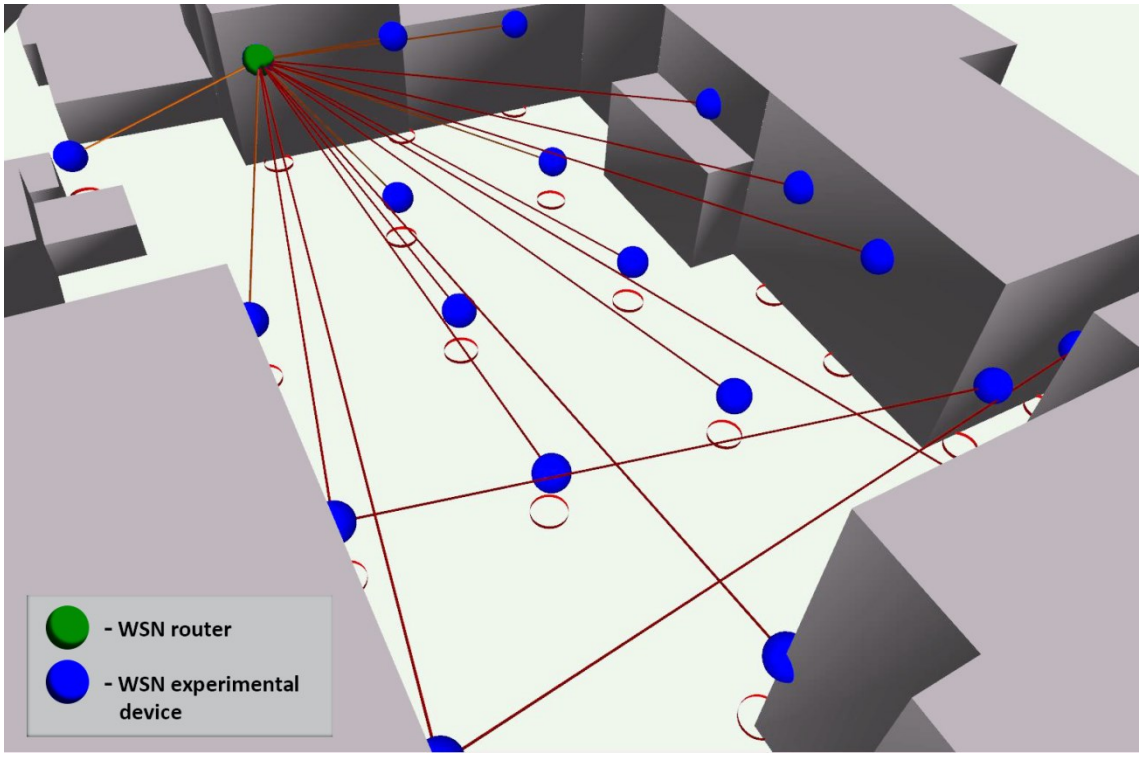

Figure 4.1: Example deployment of the ad-hoc testbed.

To provide remote testbed access from the Internet, we adopted a very lightweight approach. In our implementation the router depicted in Figure 3.2 shares a small part of the address space of the wide-area network. WSN router only forwards datagrams on the network layer, provides the Internet connectivity and does not include any gateway functionality. For making each experimental device directly addressable and to avoid the problem with lack of IPv4 address space, we used IPv6 with 6LoWPAN optimization. The alternative would be to use an application layer gateway such as used in ZigBee, ZWave, Xbee, etc. architectures [15]. However, these gateways are complex to design and manage, and would introduce an additional tier in the system architecture.

The management network enables monitoring, controlling and composing network functionality by using the CoAP protocol on top of UDP/IPv6/6LoWPAN. CoAP includes several HTTP functionalities re-designed for constrained devices such as WSN devices and it is built on top of User Datagram Protocol (UDP)[45]. Therefore it has smaller overhead and enables multicast group communication. To support all the required monitoring, control and composition functionality, we developed a set of CoAP handlers that enable wireless experimenters to remotely play with the testbed. These are detailed in Section 4.4.

For the experimental network we use the fully modular and reconfigurable CRime stack [36]. The ProtoStack [36] tool has been developed to support modular protocol development that would enable easy experimentation with multi-hop routing algorithms using also learned link characteristic for the routing decision - thus enabling cognitive networking experimentation. As ProtoStack relies on the Contiki OS, the extension has to use this operating system. For the management network we use Contiki's

6LowPAN/IPv6 stack with RPL. This complies with the architectural specifications in Section 3.3. The RPL protocol is used for automatic management network discovery and configuration. To enable two protocol stacks running in parallel, we used our previous dual-stack Contiki adaptation [10].

4.2 VESNA Platform with Dual-stack ContikiOS

The VESNA platform had a pre-existing dual radio extension board with the following options: TI CC1101 (868 MHz), TI CC2500 (2.4 GHz), AT86RF231 (2.4 GHz) and AT86RF212 (868 MHz) transceivers. Note that the extension board can support an Atmel and a TI transceiver at the same time, however it cannot support two Atmels or two TIs at the same time. For our implementation we decided to use the TI CC1101 (868 MHz) for the experimental network. This is an open and reconfigurable transceiver with a very minimal implementation on basic MAC functionalities that complies with the architectural specification in Section 3.3.1. And as second transceiver, we use the AT86RF231 (2.4 GHz) for the management network. This is an IEEE 802.15.4 compatible transceiver suitable for low power 6LoWPAN networks.

While the hardware set-up already supported dual stack (one for management and one for experimentation), a solution for a dual-stack OS had to be developed. Contiki OS includes two protocol stacks, one based on uIPv6 that can be configured as 6LowPAN/uIPv6/UDP/CoAP and the second protocol stack is custom and is referred to as Rime. The two stacks can run one at a time and not in parallel. 6LowPAN assumes an IEEE 802.15.4 compatible transceiver and since only the Atmel transceivers comply with this, the most natural decision was to consider the Atmel transceivers for the management network and the TI transceivers for the experimental network.

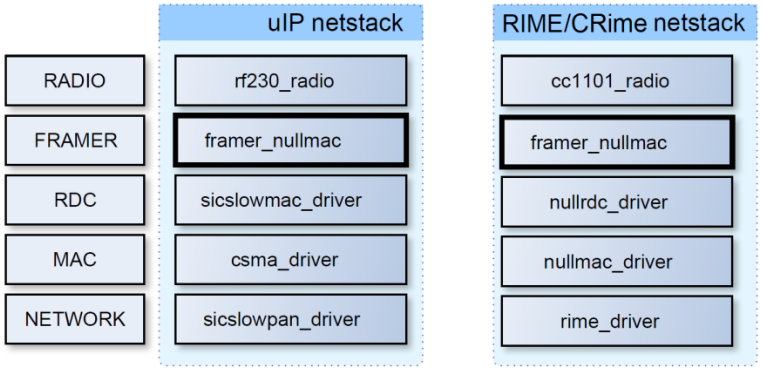

Figure 4.2: Dual stack Contiki using 6LowPAN for the management network.

Next, we extended the Contiki OS with dual stack operation. The original code uses compile-time defined network layers. Some layers are used by both Rime and uIP at the same time (see framer nullmac in Figure 4.2), so we modified the networking code to explicitly pass information about which network stack the current packet belongs to. It should be noted that in a single stack Contiki, Rime uses 2 bytes for node network address, while uIP requires 8 bytes. To keep Rime packet small, thus maintaining the low power consumption of the Rime stack, we modified Contiki to permit different network address size for Rime and uIPv6 packets respectively.

Finally, we integrated the new, composable Rime (CRime) [36] network stack that enables reconfigurable protocol stacks in the Contiki OS and configured the operating

system to support the 6LoWPAN-based management network and the CRime-based experimental network in parallel as depicted in Figure 4.2.

4.3 Software Upgrades, Updates and Reconfiguration

Software upgrades require a bootloader running on the VESNA platform and a large image to be sent to the node over the management network. In the case under investigation, the full image of the monolithic dual stack is in the range of 400 kB as shown in Table 4.1. This image corresponds to a particular application (i.e. single experiment) and it needs to be changed if the experiment requires another application (i.e. flash the node).

Table 4.1: Approach transfer size.

Approach	**transfer size [B]**	**useful size [B]**	**overhead [%]**
Monolithic dual stack image	410900	410900	0
Hello-world ELF app	1752	399	78
Trickle RIME ELF app	11316	3930	65
Monolithic dual stack config. packet	1635-7937	1635-7937	0

In order to support minor updates of drivers and applications, we used dynamic loading through the Contiki ELF (Executable and Linkable Format) loader module. In order to support dynamically reprogrammable network stacks, we investigated the possibility of transferring new stack compositions, each representing a new experiment. This requires splitting the application into two parts.

The first part, called core, is responsible for loading the minimal Contiki OS with added ELF loader functionality. This part of the node firmware is not changed during reprogramming. It is responsible for downloading the ELF application through the management network and to dynamically link it with the core OS. To implement it, we had to include the base Contiki image with uIPv6, TCP/UDP and CoAP as well as, also support for: (1) SD card driver and Contiki Coffee FS; (2) utility application to receive ELF file from the network and write it to a file; (3) ELF loader (generic and CPU architecture specific part) to do actual ELF file relocation; and (4) the symbol table which stores the addresses and names of all core OS functions, which might be called by the ELF application.

The second part is the ELF application. The application calls functions exported by the core OS, and is compiled as a standard ELF file. The relocation of ELF application as it is depicted in Figure 4.3 uses metadata within the ELF file to find the locations of unresolved data addresses, function calls, code execution jumps etc. Each such location is then overwritten with the resolved value, calculated from metadata, known locations of application sections and core function.

Part of the monolithic image from the previous approach could be loaded as an ELF application and the same result could be achieved as in OS upgrade. In this case we would need to split monolithic dual stack image into two parts, core OS (with uIP management network) and ELF application (with CRime experimental network). In addition to that we have the option to leave some code parts, used only by CRime, in the core OS. In particular, we decided to leave the TI CC radio driver in the core OS. As that particular piece of code is already stable, we expect it will not require frequent updates. This resulted in about 50% smaller ELF application file.

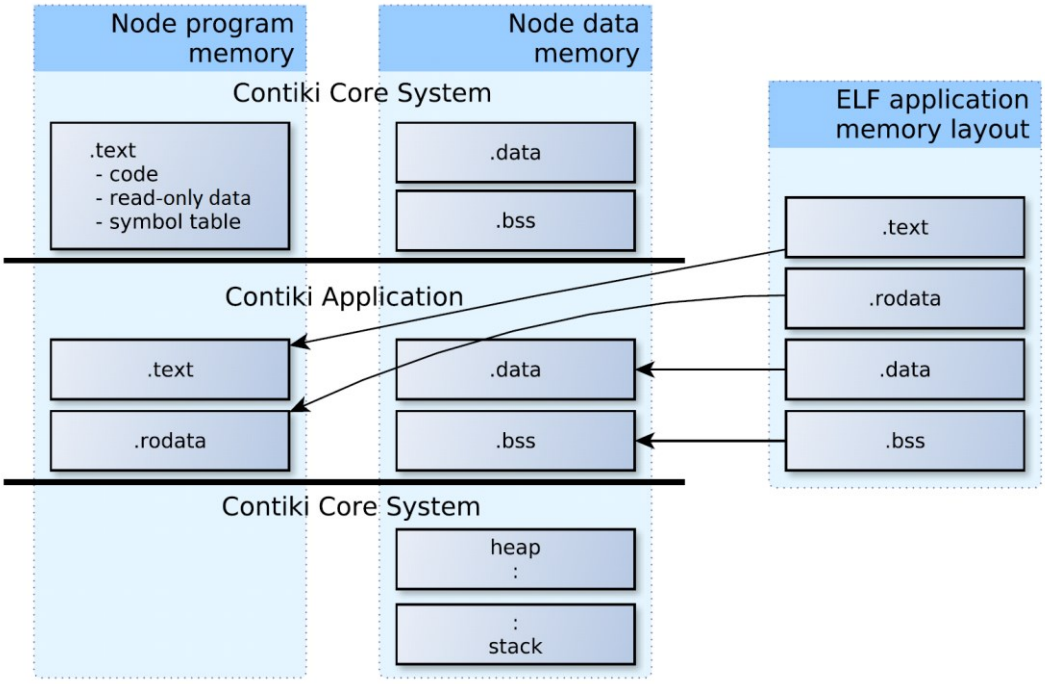

Figure 4.3: Loading ELF application.

We looked at the size of the file to be transferred to the nodes over the air for the very simple hello-world application and for a more complex trickle stack. The size of the helloworld ELF file is 1.7 kB while the size for the trickle ELF file is 11 kB as listed in Table 4.1. The trickle ELF application is small compared to the full OS image, but it still has a significant overhead due to the ELF file metadata.

4.4 CoAP Handlers Using REST Principles

All the enablers for monitoring, control and composition are implemented as CoAP handlers and developed using REST principles. This section describes the procedures that use these handlers to address the requirements in Section 3.2.

4.4.1 Remote Monitoring and Diagnosis

Figure 4.4 presents a sequence diagram that uses CoAP handlers for remote monitoring and diagnosis to perform device hardware and experiment monitoring. First, the */hardware/status* is used to receive transceiver configuration setup such as current channel, transmit power, sampling rate, etc. Then the node sends the settings to the user. Second, after the experiment is started, */crime/stack* push is used to receive periodic status messages of the running experiment containing the values of LQI and RSSI of the last received packet. Then, after the completion of the experiment, */crime/get_prr* is used to retrieve PRR of the overall experiment. The node then sends this data to the user. As the last event, using the */crime/get_results*, the experimenter will retrieve the logged results (RSSI, LQI), stored on the SD card. The node will send the data to the user in chunks if the size of the data exceeds the maximum size of a packet payload.

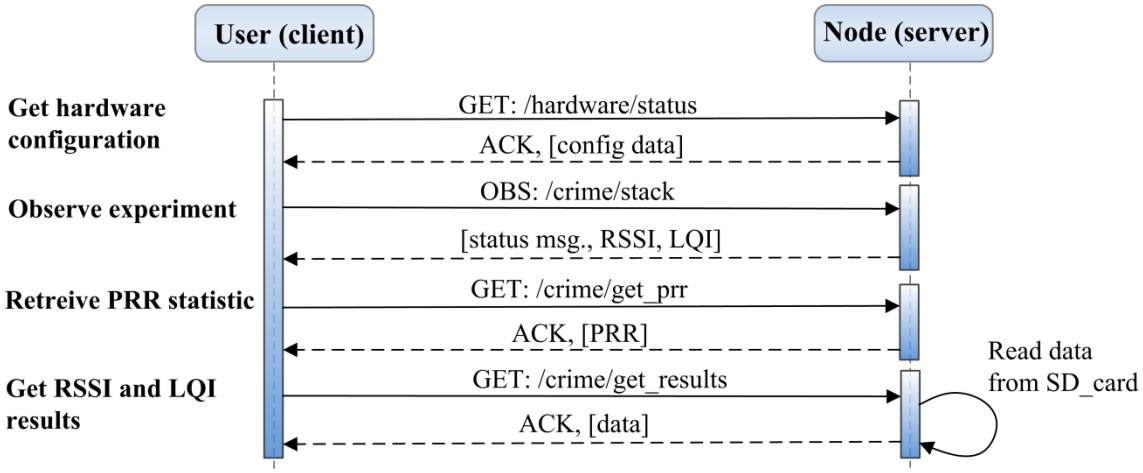

Figure 4.4: Remote monitoring.

4.4.2 Remote Parameter Tuning

For remote parameter tuning we implemented several CoAP handlers. Figure 4.5 presents a sequence diagram that uses two of them to adjust the channel and transmission power of transceivers. First, the */ti_cc/set_power* is used with the desired power level as a parameter. The experimenter (script) sends it to the target node which acknowledges the request. Then, the */ti_cc/set_chn* is sent with a numeric value specifying the desired channel. The node acknowledges also this request. Third and last, using the */ti_cc/restart* handler triggers the reboot of the transceiver. After the reboot, the new values, which have been written in the appropriate registries by the routines called by the CoAP handlers, take effect, thus appropriately configuring the transceiver.

In our implementation, several parameter tuning handlers have also been developed for the management network. Instead of referring to the TI transceiver, the requests refer to the Atmel transceiver. This is implemented in a RESTful architecture by changing */ti/* into */atmel/* in the CoAP request.

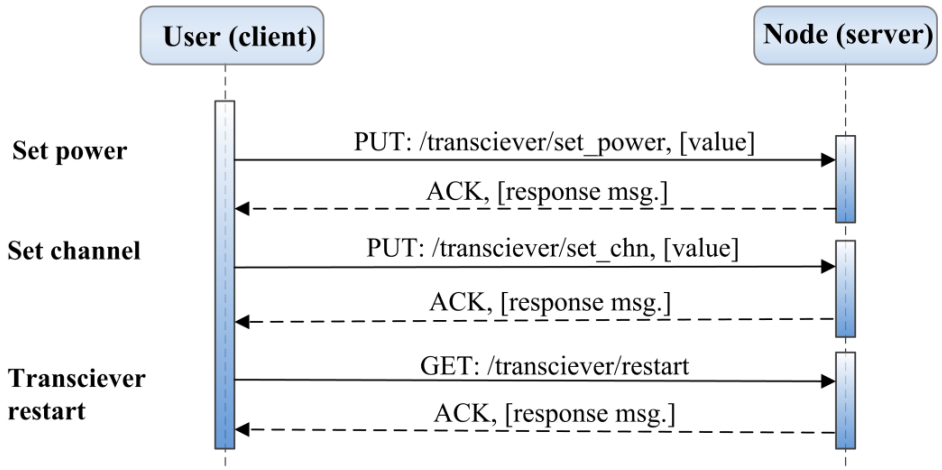

Figure 4.5: Remote configuration of channel and transmission power.

4.4.3 Over the Air Software Updates and Upgrades

Figure 4.6 presents a sequence diagram of CoAP handlers for over the air software updates and upgrades to perform a full operating system (OS) update, and Figure 4.7 presents a sequence diagram that uses the same CoAP handlers for driver and application updates.

First, in Figure 4.6, the */firmware/card_*format is used if users want to wipe any existing data from the SD card. The node acknowledges this request. Second, the */firmware/header* is used to upload a new OS image metadata such as size and CRC32 of the image and slot ID where image will be stored on the card. The node also acknowledges this request. Third, the */firmware/upload* is used to upload the new OS image. The node acknowledges each transferred chunk of data.

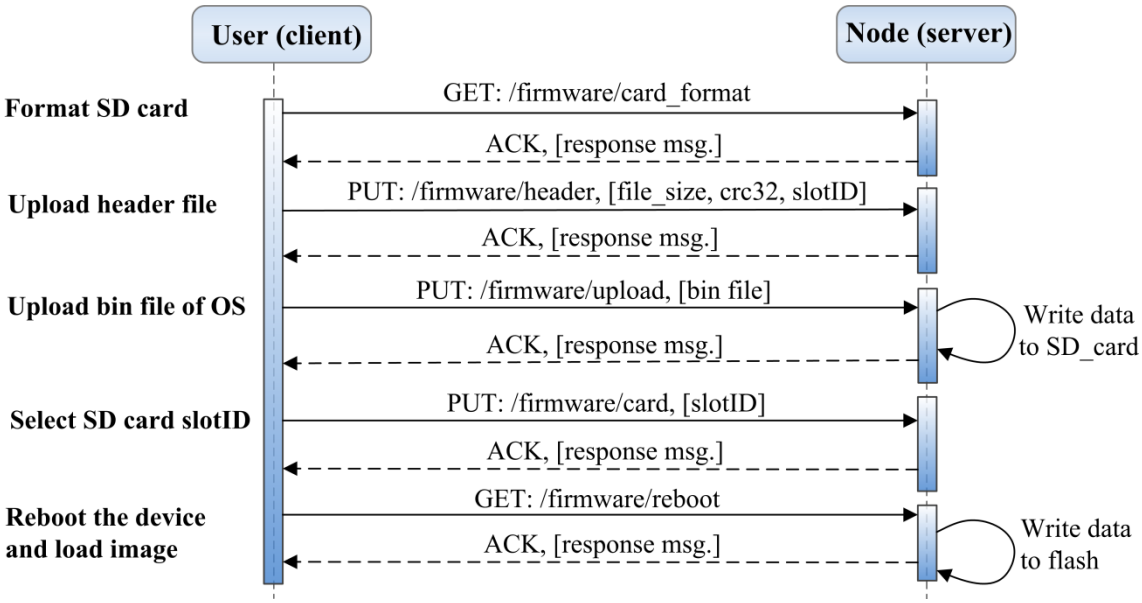

Figure 4.6: Full operating system/firmware upgrade.

Fourth, the */fimware/card* slot is used to select the memory slot of the SD card from which the new OS image has to be loaded. The node will also acknowledge this request. And fifth, the */firmware/reboot* is used to reboot the device and start the loading process of the OS image from the SD card into the flash.

The first operation in Figure 4.7 uses the */elf/header* handler that uploads the application meta-data such as file size, CRC32 and application name. The node acknowledges this request. Second, the */elf/upload* is used to upload the application file. The node acknowledges each transferred chunk of data. After the relocation is done, the node sends the application ID. Third and fourth, the */elf/start* or */elf/stop* is used with application ID as parameter to start or stop the application. The node acknowledges also these requests.

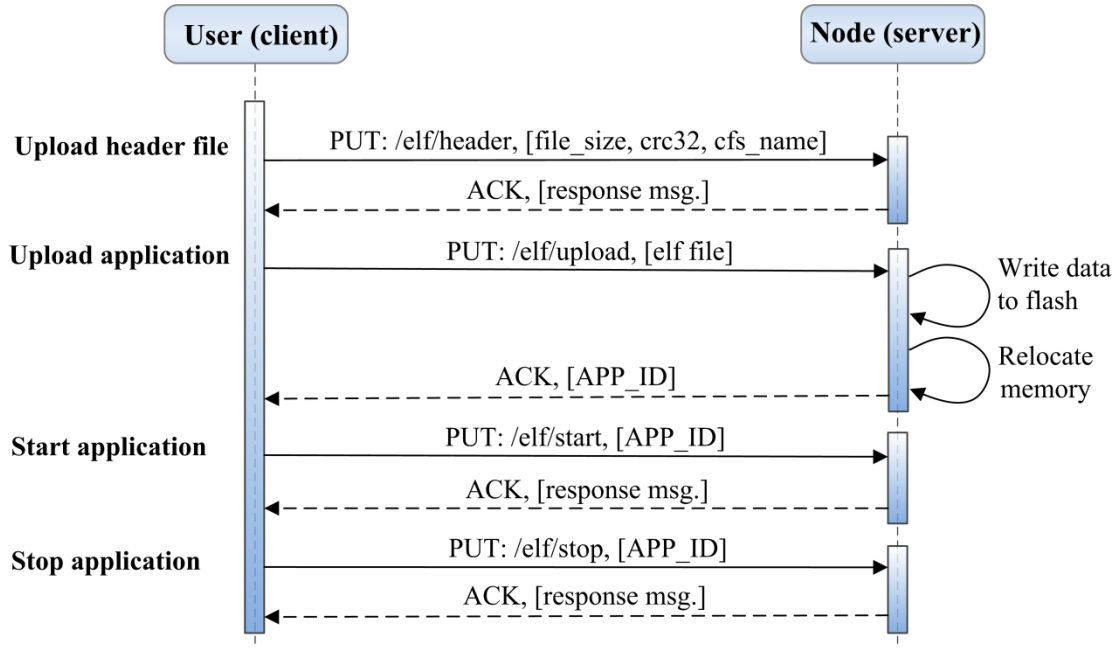

Figure 4.7: Application or drivers update.

4.4.4 Modular Stack Reconfiguration

Figure 4.8 presents a sequence diagram of CoAP handlers for the modular stack reconfiguration to perform runtime modular stack reconfiguration using JSON configuration messages. First, the */crime/upload* is used to upload the JSON stack configuration message. The node acknowledges successful upload. Second, the */crime/init* is used to initialize the stack. The node acknowledges the request after the stack is initialized. Third, the */crime/start* handler is used to trigger experiment with the desired sample rate and number of packets that should be transmitted. The node also acknowledges this request. Finally, as fourth, the */crime/stop* is used to stop the running experiment.

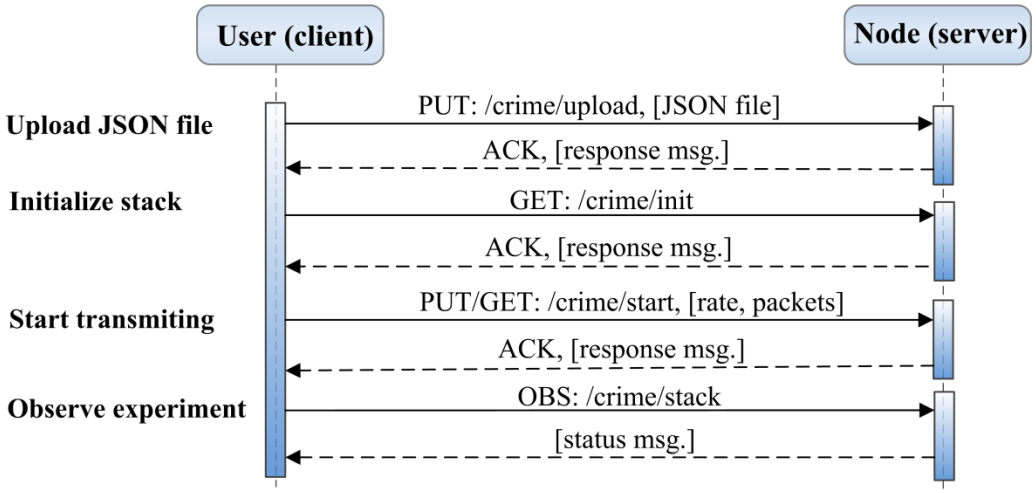

Figure 4.8: Run-time stack reconfiguration.

Chapter 5

Performance Evaluation of WSN Testbed

In this chapter we introduce architecture validation and evaluation. First we start with evaluation of radio transceivers for the wireless management network. Next, the usage of experimental testbed is demonstrated by performing two sets of experiments, one for the evaluation of the management network and the second for the evaluation of the experimental network. Likewise, other types of experiments can also be developed and performed on the infrastructure, but they are beyond the scope of this thesis.

5.1 Evaluation of Radio Transceivers for the Management Network

For wireless network design it has been shown [46] that the reception probability of the nodes in the network has three regions. In the first region, also called the effective region, the packet success rate is above 90%. In the second region, also referred to as transitional region, the packet success rate falls off smoothly but exhibits high variation. In the third region, also referred to as clear region, the packet success rate is below 10%. The boundaries of the three regions and the fall of the success rate are determined by several factors, among which are the frequency band and the used transceiver.

In Section 4.2, we narrowed down the candidate transceivers supporting the management network to two IEEE 802.15.4 Atmel transceivers: AT86RF231 (2.4 GHz) and AT86RF212 (868 MHz). The first step in evaluating these transceivers was to determine the three operating regions. The experiment was carried out in a 55 meters long corridor of a long building where several WiFi access points are also active. Figure 5.1 plots the three regions empirically determined in our experiments for the AT86RF231 (2.4 GHz) transceiver. The tests show that the effective region goes up to 22 m in the line of sight conditions (no obstacles in the corridor). Additionally, we performed experiments to understand how the throughput is affected by the packet rate as shown in Figure 5.2. The results show that rates exceeding 70 packets per second lead to packet losses.

The plots for the reception success rate as a function of distance for the AT86RF212 (868 MHz) transceiver are depicted in Figure 5.3, with the effective region ending at 28 m. Due to the hardware limitation, with this transceiver we managed to achieve application rate of 41 packets per second with 100% of packet success rate.

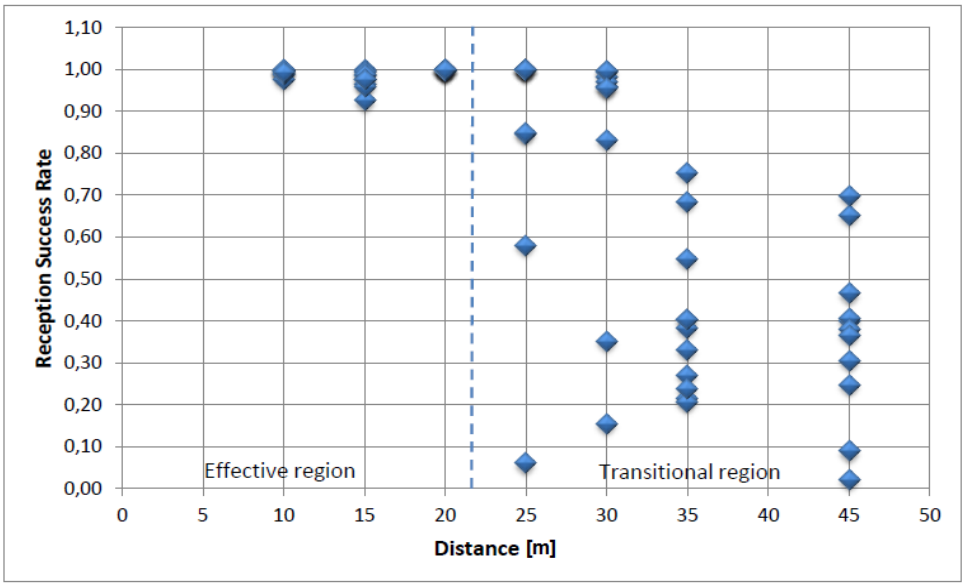

Figure 5.1: Reception success rate as a function of a distance for the AT86RF231 (2.4 GHz) transceiver, with application rate of 70 packet/s.

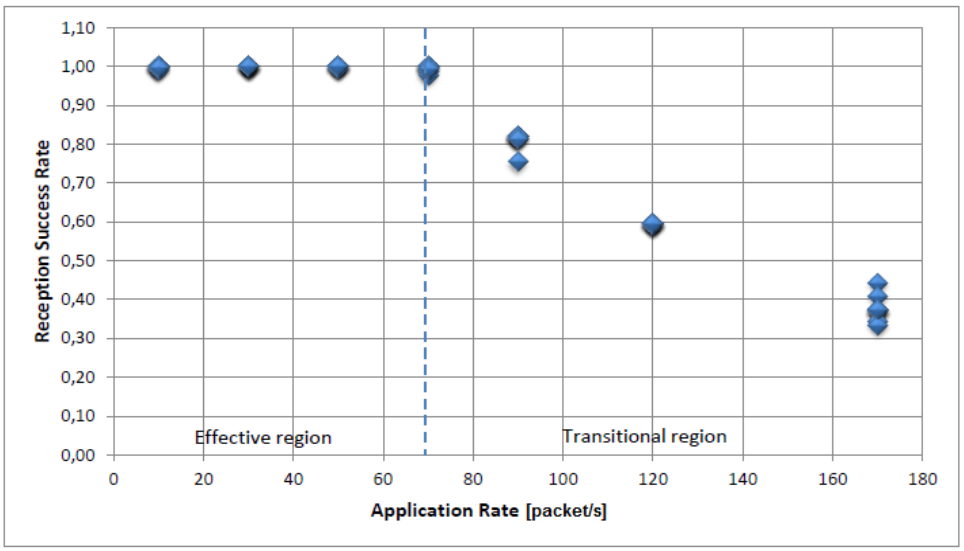

Figure 5.2: Reception success rate as a function of application packet rate for the AT86RF231 (2.4 GHz) transceiver, at the distance of 16 m.

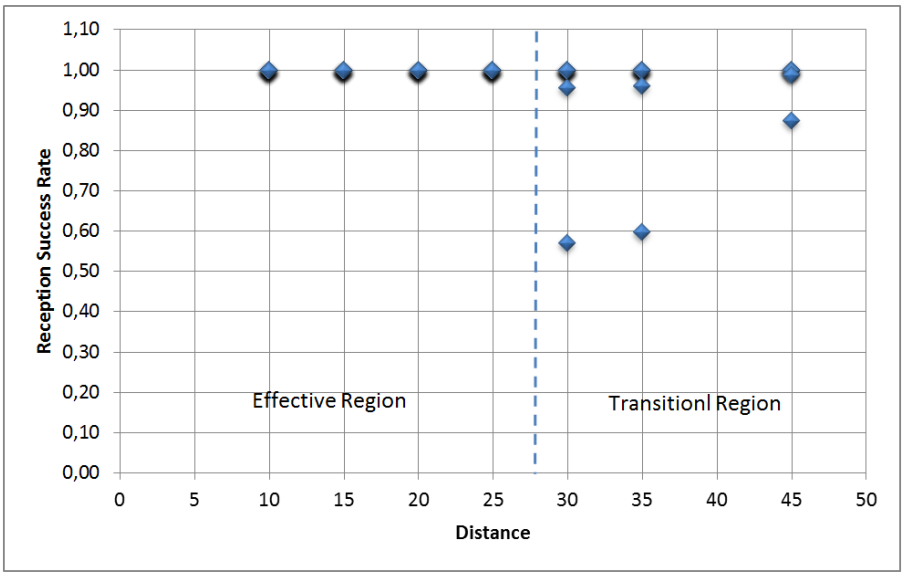

Figure 5.3: Reception success rate as a function of a distance for the AT86RF212 (868 MHz) transceiver, with application rate of 40 packet/s.

After determining the effective region for the two transceivers under consideration, we looked at the application transfer rates and various settings that influence these. For instance, by using the header compression enabled by 6LoWPAN, the application payload can be increased thus maximizing the data rate. Figure 5.4 presents the experimental setup used for measuring the uplink/downlink throughput. The CoAP client was mimicking reprogramming functionality by sending large files for reprogramming the nodes running CoAP server and data collection functionality by requesting (randomly generated) data from the nodes. The CoAP clients are located on a wired IPv4/IPv6 network, then use a gateway towards the border router which has a wireless management interface for the nodes. The links between the nodes and the border router were within the effective regions, hence reliable.

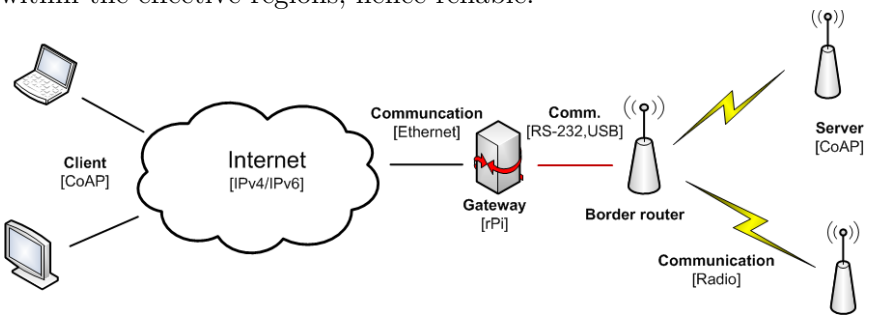

Figure 5.4: Experiment setup.

We performed two different types of experiments, upload and download for two different sets of radio transceivers Atmel AT86RF212 at 868 MHz (Table 5.1) and AT86RF231 at 2.4 GHz(Table 5.2). Each type of the experiment had five different steps and with each step we were increasing the size of the data (i.e. file) to be transmitted from 128 to 256000 bytes. To get more reliable results we repeated each step 10 times and then we calculated the average throughput value. With respect to the upload and download we performed 100 measurements per radio transceiver. From the results in

Tables 5.1 and 5.2, it can be seen that AT86RF231 is achieving higher data throughput and the links are more stable.

In our evaluation we only considered packets that contain payload data, avoiding acknowledgment messages that are sent for each packet and that are not relevant for the application data rate. We decided to use HC01 and HC02 compression for 6LoWPAN because if the CoAP client is accessing the testbed from a different network subnet, the IPv6 address will not be fully compressed in any case. Packet fragmentation was disabled. MAC header compression was not used, because it is supported only by the AT86RF212 MAC. As a note, there are several configurations and tunings that can be performed with such an evaluation. Configurations in the Contiki OS, the used drivers and the point from which the client is accessing the node influence the final performance of the wireless management network.

Table 5.1: Upload and Download troughput of AT86RF212.

Size [bytes]	Download time [seconds]	Upload time [seconds]	Download throughput [kbps]	Upload throughput [kbps]
128	0.185	0.167	5.405	5.988
1280	1.705	1.655	5.865	6.042
12800	16.614	16.666	6.019	6.000
128000	176.324	168.800	5.671	5.924
256000	346.699	339.327	5.768	5.894

Table 5.2: Upload and Download troughput of AT86RF212.

Size [bytes]	Download time [seconds]	Upload time [seconds]	Download throughput [kbps]	Upload throughput [kbps]
128	0.086	0.069	11.627	14.492
1280	0.686	0.694	14.577	14.409
12800	6.673	6.965	14.985	14.357
128000	68.191	68.899	14.664	14.513
256000	139.696	139.241	14.316	14.363

Tables 5.1 and 5.2 are summarizing results where packets with 64 bytes of application payload have been used. To analyze different approaches described in Section 3.3.2, for instance to transfer 256000 bytes, which could correspond for instance to a new firmware image for full OS reprograming, 140 seconds are needed when using AT86RF231 and 347 seconds when using AT86RF212. For the dynamic loading of a simple application or run-time reconfiguration using a non-optimal JSON format is the image of 1000 bytes, less than a second is needed with AT86RF231 and under 2 seconds with AT86RF212 - thus being more economic for the wireless management network.

5.2 Evaluation of the Wireless Management Network

The wireless management network of the testbed will have to provide multi-hop communication to reach all the nodes and ensure a well-connected network so that all nodes can be reached for control, reconfiguration, data collection and software upgrade purposes as discussed in Section 3. According to the existing literature, characterizing a multihop wireless network and designing the desired management network is not a trivial

task and some links may as well operate beyond the effective region, i.e. in the transitional region.

With respect to the transitional region, it has been shown [47] that (1) the link quality is not correlated with distance, and (2) the extent of the transitional region seems to depend on the environment (e.g. outdoor, indoor, presence of obstacles), and the radio hardware characteristics. Additional observations made by the same authors that are particularly relevant for this evaluation: (1) link quality is anisotropic; (2) links with very low or very high average PRRs (Packet Reception Ratio) are more stable than links with moderate average; (3) over short time spans, links may experience high temporal correlation in packets reception, which leads to short periods of 0% PRR or 100% PRR; (4) the colocation of 802.15.4 and 802.11b networks affects transmission in both networks due to interference, but the transmission in 802.11b networks is less affected; (5) the co-location of IEEE 802.15.4 and 802.15.1 (Bluetooth) networks affects mostly the transmissions in the IEEE 802.15.4 network; and (6) the colocation of IEEE 802.15.4 networks and domestic appliances can significantly affect the transmission in the IEEE 802.15.4 networks.

In order to realize a wireless management network that services well the testbed by being able to reach all nodes at any time, providing good throughputs to enable configuring, controlling and upgrading the network, it would be highly desirable that as many as possible of the links forming the network are in the effective region. For the ones belonging to the transitional region, it is desirable that they are connected to the core of the network via more than one (highly varying) link. In order to achieve this, the candidate transceivers forming the network have to be evaluated and the operating environment and topology also need to be taken into consideration.

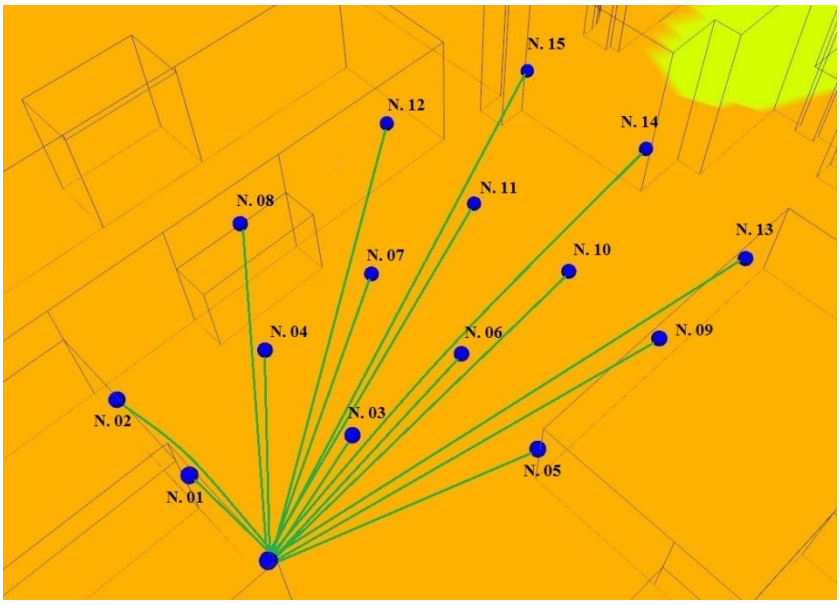

Figure 5.5: Network topology of managment network.

For the validation purposes of management network we decided to perform uplink/downlink experiments on the deployed WSN experimental testbed that use AT86RF231 transceiver at 2.4 GHz for the management network, described in Section 4.1. WSN experimental nodes were configured with IPv6 address, CoAP protocol and each node was reachable on a single-hop link from WSN sink-router, as it is depicted in Figure 5.5. As a testing client application we developed a Python script using CoAP

library that was performing uplink/downlink experiments with each node in the network, and calculating the transmission time and the throughput.

Table 5.3: Uplink/downlink data rate of 15 Nodes.

Payload	**1024 B**		**10240 B**		**102400 B**		**256000 B**		**409600 B**	
Node	Uplink [kbps]	Downlink [kbps]	Uplink [kbps]	Downlink [kbps]	Uplink [kbps]	Downlink [kbps]	Uplink [kbps]	Downlink [kbps]	Uplink [kbps]	Downlink [kbps]
Node 01	9,40	9,22	8,86	9,13	9,19	9,06	8,54	8,83	8,92	9,01
Node 02	9,42	9,06	9,50	9,18	9,28	8,95	9,30	9,01	9,32	9,05
Node 03	9,58	9,18	9,64	9,25	9,52	9,07	9,59	9,20	9,36	9,14
Node 04	9,50	9,22	8,46	9,28	9,44	9,27	9,40	9,05	9,35	9,18
Node 05	9,48	9,11	9,59	8,67	9,60	9,02	9,53	8,93	9,11	9,10
Node 06	9,57	9,23	9,61	9,25	8,92	9,11	9,29	9,23	9,40	9,14
Node 07	9,66	9,21	9,61	9,21	9,54	9,06	9,37	8,93	9,47	9,21
Node 08	9,52	9,12	9,59	8,76	9,47	8,95	9,42	9,14	9,46	9,04
Node 09	9,46	9,09	9,54	9,16	9,37	8,96	9,15	8,88	8,91	8,65
Node 10	7,97	9,11	7,25	9,00	7,98	8,93	8,58	9,06	8,43	8,98
Node 11	8,64	8,21	6,30	8,60	8,96	8,64	8,77	8,84	8,62	8,78
Node 12	9,56	9,17	9,59	9,24	9,35	9,14	9,51	9,12	9,55	9,07
Node 13	9,20	9,05	8,88	9,19	8,95	9,08	8,61	9,02	8,89	9,00
Node 14	9,11	8,86	9,27	8,31	8,64	8,71	8,33	8,58	8,30	8,92
Node 15	8,99	6,87	8,55	8,42	8,17	8,15	6,32	6,74	6,14	6,51

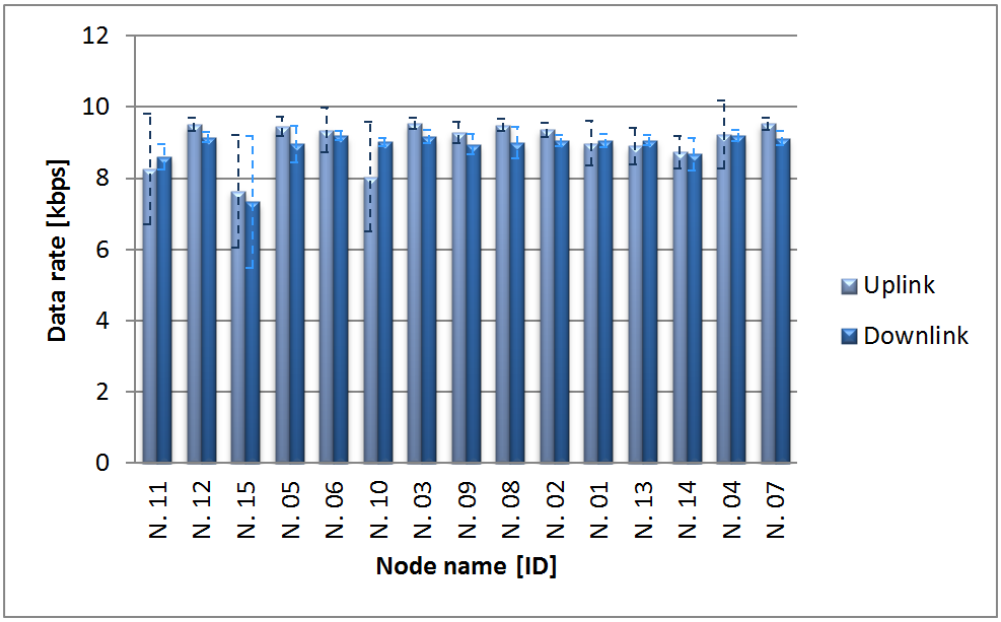

Figure 5.6: Average uplink/downlink data rate with standard deviation of 50 experiments per node.

The experiment had five different steps and with each step we were increasing the size of the data (i.e. file) to be transmitted between 1280 and 409600 bytes. In order to get more reliable results, each step was repeated 5 times and mean average was calculated shown in Table 5.3, resulting in total of 50 experiments per node. Total average uplink/downlink data rate with standard deviation for 15 selected nodes are shown in Figure 5.6. We can see that for some nodes such as *Node 15* we are getting lower rates because of the interfering objects that are between the router and the node.

Table 5.4 presents the sizes of selected files or messages that have to be transferred over the air for reprogramming or reconfiguring the network. For full OS reprogramming, the total number of bytes to be sent over the air is 410,900. If we only send a partial code update, the ELF file for the trickle application in this case, the total transfer size is 11,316 bytes. An average message (i.e. CoAP handler) for transceiver reconfiguration requires transferring 32 bytes while a JSON message for reconfiguring the entire protocol stack also amounts to transferring several thousands of bytes, in this case, 7937 bytes.

Table 5.4: Transfer size.

Use case	**Transfer size [byte]**
Monolithic dual stack image	410900
Trickle RIME ELF app	11316
Transceiver reconfiguration	32
Modular stack reconfiguration	7937

The measured average data rate and packet loss of 15 selected nodes in the testbed was 9,02 kbps and 7.054 % of packets, respectively, between the nodes that are 10-30 meters apart as the case in our deployment. Based on these values, the second column in Table 5.5 shows the average transfer times required for the messages listed in Table 5.4. The last column of the table presents node local measurements of the relocation time necessary to find the locations of unresolved function or data addresses, function calls, code execution jumps, etc. As expected, the results confirm that both transfer and relocation times for software upgrades are significantly larger than for reconfiguration messages. Furthermore, while the transfer time for full OS update is larger than for partial code updates, the relocation is smaller for the first (68 sec) versus the second (239 sec). If we consider transfer and relocation time in the context of setting up a testbed for subsequent experimenting, it can be seen that preparation of an experiment can take from a few seconds to a few minutes, depending on the complexity of the experiment which, in most cases, is proportional to the number or required configuration messages.

Table 5.5: Application deployment time.

Use case	Transfer time [s]	Relocation time [s]
Monolithic dual stack image	355	68
Trickle RIME ELF app	9	239
Transceiver reconfiguration	<1	<1
Modular stack reconfiguration	6	<3.0

5.3 Evaluation of Experimental Network

As an example used to evaluate the experimental network of the deployed testbed, we decided to perform RSSI-based Rings Overlap Localization algorithm [48]. Localization algorithms can be divided into two categories: range-based localization technology and range-free localization technology. Range-based localization depends on the assumption that the absolute distance between a sender and a receiver can be estimated by received signal strength or by the time-of-flight of the wireless signal from the sender to the receiver. While range-free algorithms depend on proximity sensing or connectivity information and they never try to estimate the absolute point-to-point distance based on received signal strength. The selected algorithm is based on range-based technology and the goal of this algorithm is that each WSN node the (receiver/transmitter) uses overlapping rings to narrow down the possible area in which transmitter/receiver is located.

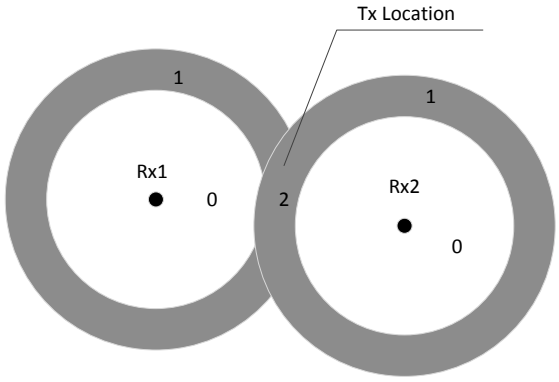

Figure 5.7: Transmitter localization using Rings Overlap localization algorithm.

Each receiver that is included in the algorithm, based on the received strength of the RSSI signal, calculates the possible distance at which the transmitter is located. As it is depicted in Figure 5.7, as a result of the algorithm we obtain a ring around each receiver where the estimation of the transmitter location increases from zero to one. With each additional intersection of rings, the estimated location of transmitter increases by one.

In our experimental scenario using the CRime protocol stack we decided to configure simple broadcast stack on the node *N. 10*. In the experiment we will refer to this node as the transmitter. On the other hand, we developed a Python script using the CoAP library that could subscribe to the observable handlers of the nodes *N. 06, N. 07, N. 11* and *N. 13* and log the RSSI measurements of each received packet. We will refer to these nodes as receivers. After we successfully managed to collect RSSI measurements using CoAP handlers described in Section 4.4.1, we used a MATLAB script to create Figure 5.8 depicting rings of estimated location. As it can be seen using just RSSI data and Gauss Kruger coordinates of each receiver, we got the estimation of transmitter location with an error of 7.5 meters.

As the next step we decided to try to improve the estimated location. With the same data that had collected in the first step we used additional information such as antenna pattern and surrounding obstacles that were interfering the received signal. As a result that is depicted in Figure 5.9, we managed to reduce the estimation error from 7.5 to 1.1 meters.

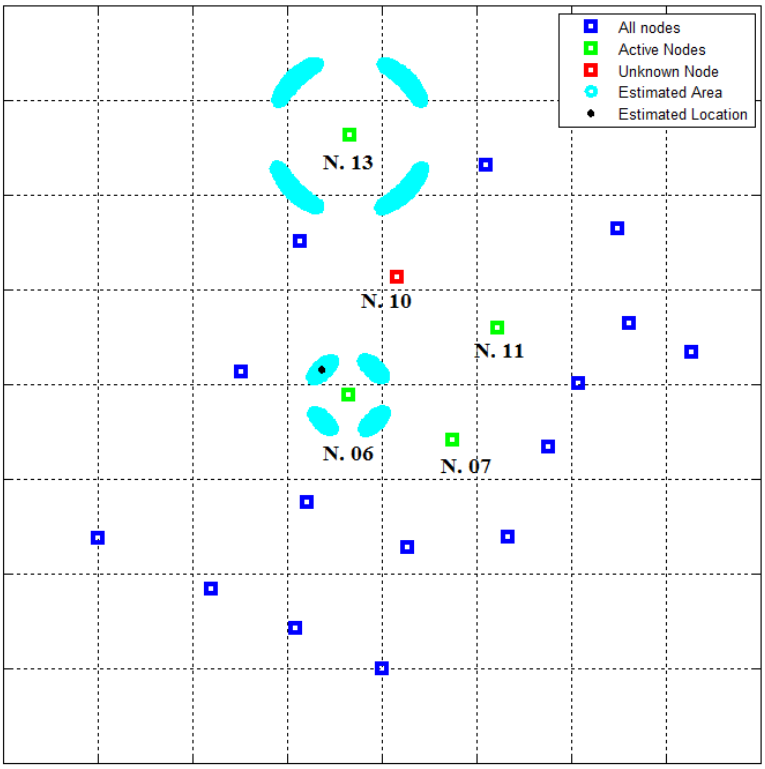

Figure 5.8: Localization of transmitter using only RSSI data, no knowledge about radio environment (error~7.5m).

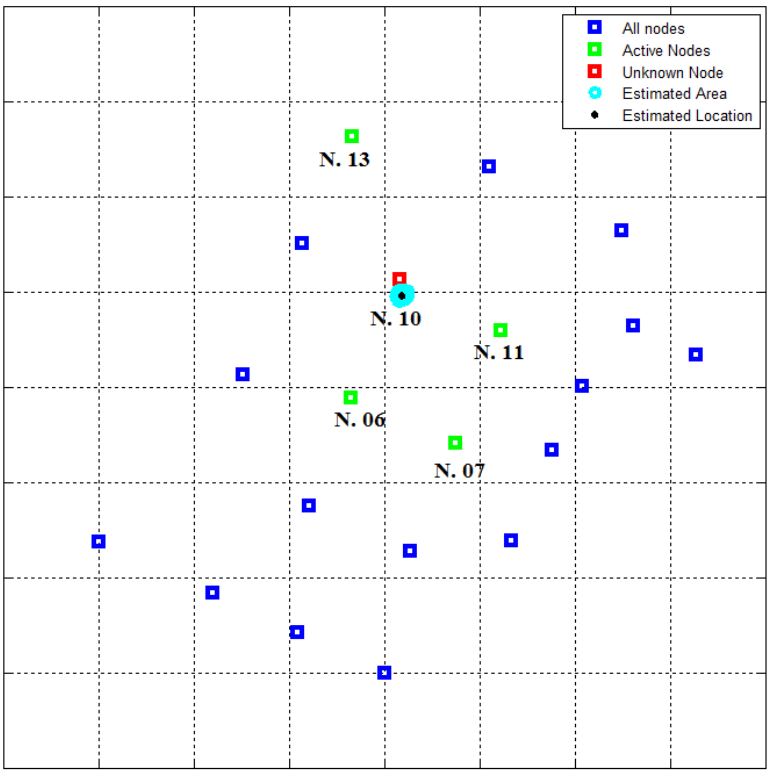

Figure 5.9: Transmitter localization using RSSI with some knowledge of radio environment (error~1.1m).

Chapter 6

Conclusions

In this thesis, we argue that an architecture for fully reconfigurable wireless sensor network testbed, suitable for fast on-site deployment, demonstration and testing in real operating environment, can foster the adoption of wireless sensor networks in industrial and building automation environments. These environments pose specific challenges that we translated to concrete requirements and design goals that a testbed has to fulfil with respect to remote operation in a user-friendly way. In summary, such testbed should be comprised of WSN nodes supporting configurable transceiver(s), configurable and possibly modular protocol stack(s), and a monitoring, control and composition block; furthermore, they should include support for remote software and firmware upgrades and reconfigurations. To this end, different modes of remote programming / configuration may be applicable to different types of experimentation / operation, also depending on the basic design of the sensor nodes and their capabilities. As a result to the list of requirements, we propose experimental architecture for a single-tier WSN testbed for experimentally driven research and development that can be easily deployed on an ad-hoc basis in any target environment and remotely configured and controlled using simple RESTful APIs.

In the reference implementation we presented the operation of developed CoAP handlers for monitoring, control and composition. The implemented dual stack network enables to remotely upgrade the nodes of the testbed without the need for wired infrastructure and still benefit from a management network that is separated from the experimental network rather than piggybacking on it. The improved application throughput, together with the smaller application level updates / reconfiguration size significantly shorten the time required to set up a new experiment

We validated and evaluated the operation of management and experimental networks in terms of achieved average data rate, packet loss ratio and application deployment time under different modes of supported remote reprogramming / reconfiguration. In particular, we showed that in some cases a trade-off needs to be carefully considered between the time needed for the transfer of a new firmware image and the relocation time for the software upgrade.

6.1 Future Work

While we achieved all the initially set goals of the thesis, we present in this section some guidelines for possible future work and improvements of the proposed architecture, the deployed WSN testbed and for future experiments.

The first step in improving the proposed architecture would be in complementing the existing set of RESTful CoAP handlers with some that would simplify monitoring and control of more sophisticated experiments.

Regarding the deployed experimental testbed, uplink and downlink throughput of the management network could be significantly improved by employing the HC01 and HC02 packet compressions that are already part of Contiki OS. With these compressions we could in theory improve the throughput by 30-40 % compared to results obtained in the experiment described in Section 5.2.

Future work on the experimental side could improve the existing localization experiment, described in Section 5.3, to support real-time localization of a transmitter or a receiver. Probably more important, though, we are planning to adopt few static and mobile routing algorithms for the CRime library that will exploit cross-layer RSSI, LQI and PRR information for finding the best single-hop and multi-hop paths that can subsequently be used in designing and executing new types of experiments.

References

[1] A. Gluhak, S. Krco, M. Nati, D. Pfisterer, N. Mitton, and T. Razafindralambo, "A survey on facilities for experimental internet of things research," *Communications Magazine, IEEE,* vol. 49, pp. 58-67, 2011.

[2] L. Sánchez, V. Gutiérrez, J. A. Galache, P. Sotres, J. R. Santana, J. Casanueva, *et al.*, "SmartSantander: Experimentation and service provision in the smart city," in *Wireless Personal Multimedia Communications (WPMC), 2013 16th International Symposium on*, 2013, pp. 1-6.

[3] J. Bers, A. Gosain, I. Rose, and M. Welsh, "Citysense: The design and performance of an urban wireless sensor network testbed," in *JJ Proceedings of the 2008 IEEE International Conference on Technologies for Homeland Security, Waltham, MA. Citeseer*, 2008.

[4] T. Šolc, C. Fortuna, and M. Mohorčič, "Low-cost testbed development and its applications in cognitive radio prototyping," in *Cognitive Radio and Networking for Heterogeneous Wireless Networks*, ed: Springer, 2015, pp. 361-405.

[5] A.-S. Tonneau, N. Mitton, and J. Vandaele, "A Survey on (mobile) wireless sensor network experimentation testbeds," in *Distributed Computing in Sensor Systems (DCOSS), 2014 IEEE International Conference on*, 2014, pp. 263-268.

[6] O. Fambon, E. Fleury, G. Harter, R. Pissard-Gibollet, and F. Saint-Marcel, "FIT IoT-LAB tutorial: hands-on practice with a very large scale testbed tool for the Internet of Things," *10èmes journées francophones Mobilité et Ubiquité, UbiMob2014,* 2014.

[7] I. Chatzigiannakis, S. Fischer, C. Koninis, G. Mylonas, and D. Pfisterer, "WISEBED: an open large-scale wireless sensor network testbed," in *Sensor Applications, Experimentation, and Logistics*, ed: Springer, 2010, pp. 68-87.

[8] R. T. Fielding, "Architectural styles and the design of network-based software architectures," University of California, Irvine, 2000.

[9] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *Communications magazine, IEEE,* vol. 40, pp. 102-114, 2002.

[10] J. Cinkelj, M. Sterk, A. Bekan, M. Mohorcic, and C. Fortuna, "Design trade-offs for the wireless management networks of constrained device testbeds," in *Wireless Communications Systems (ISWCS), 2014 11th International Symposium on*, 2014, pp. 245-250.

[11] W. W. Dargie and C. Poellabauer, *Fundamentals of wireless sensor networks: theory and practice*: John Wiley & Sons, 2010.

[12] J. Zheng and M. J. Lee, "A comprehensive performance study of IEEE 802.15. 4," ed: IEEE Press book Los Alamitos, 2004.

[13] Z. Shelby and C. Bormann, *6LoWPAN: The wireless embedded Internet* vol. 43: John Wiley & Sons, 2011.

[14] O. Gasser, "TCP/IP communication in a WSN," *Sensor Nodes–Operation, Network and Application (SN),* vol. 75, 2011.

[15] J. W. Hui and D. E. Culler, "IPv6 in low-power wireless networks," *Proceedings of the IEEE,* vol. 98, pp. 1865-1878, 2010.

[16] R. T. Fielding and R. N. Taylor, "Principled design of the modern Web architecture," *ACM Transactions on Internet Technology (TOIT),* vol. 2, pp. 115-150, 2002.

[17] M. Kovatsch, S. Duquennoy, and A. Dunkels, "A low-power CoAP for Contiki," in *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*, 2011, pp. 855-860.

[18] K. Hartke. (2014). *Observing resources in coap.* Available: https://tools.ietf.org /html/draft-ietf-core-observe-06

[19] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and scalable simulation of entire TinyOS applications," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, 2003, pp. 126-137.

[20] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki-a lightweight and flexible operating system for tiny networked sensors," in *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, 2004, pp. 455-462.

[21] A. Dunkels, "Rime—a lightweight layered communication stack for sensor networks," in *Proceedings of the European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session, Delft, The Netherlands*, 2007.

[22] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, *et al.*, "MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms," *Mobile Networks and Applications,* vol. 10, pp. 563-579, 2005.

[23] A. Eswaran, A. Rowe, and R. Rajkumar, "Nano-rk: an energy-aware resource-centric rtos for sensor networks," in *Real-Time Systems Symposium, 2005. RTSS 2005. 26th IEEE International*, 2005, pp. 10 pp.-265.

[24] MEMSIC, "Datasheet TelosB, Available online: http://www.memsic.com," 6020-0094-03 Rev A ed: MEMSIC Inc., 2013.

[25] C. Buschmann and D. Pfisterer, "iSense: A modular hardware and software platform for wireless sensor networks," *6. Fachgespräch Sensornetzwerke,* p. 15, 2007.

[26] MEMSIC, "Datasheet MICAz, Available Online: http://www.memsic.com," *San Jose, California,* 2006.

[27] G. Werner-Allen, P. Swieskowski, and M. Welsh, "Motelab: A wireless sensor network testbed," in *Proceedings of the 4th international symposium on Information processing in sensor networks*, 2005, p. 68.

[28] S. Bouckaert, W. Vandenberghe, B. Jooris, I. Moerman, and P. Demeester, "The w-iLab. t testbed," in *Testbeds and Research Infrastructures. Development of Networks and Communities*, ed: Springer, 2011, pp. 145-154.

[29] V. Handziski, A. Köpke, A. Willig, and A. Wolisz, "TWIST: a scalable and reconfigurable testbed for wireless indoor experiments with sensor networks," in *Proceedings of the 2nd international workshop on Multi-hop ad hoc networks: from theory to reality*, 2006, pp. 63-70.

[30] R. N. Murty, G. Mainland, I. Rose, A. R. Chowdhury, A. Gosain, J. Bers*, et al.*, "Citysense: An urban-scale wireless sensor network and testbed," in *Technologies for Homeland Security, 2008 IEEE Conference on*, 2008, pp. 583-588.

[31] M. Mohorcic, M. Smolnikar, and T. Javornik, "Wireless sensor network based infrastructure for experimentally driven research," in *Wireless Communication Systems (ISWCS 2013), Proceedings of the Tenth International Symposium on*, 2013, pp. 1-5.

[32] A. Bekan, J. Cinkelj, M. Mohorcic, and C. Fortuna, "An architecture for fully reconfigurable plug-and-play wireless sensor network testbed.," presented at the IEEE Globecom, San Diego, California, USA, 2015.

[33] V. Shnayder, M. Hempstead, B.-r. Chen, G. W. Allen, and M. Welsh, "Simulating the power consumption of large-scale sensor network applications," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, 2004, pp. 188-200.

[34] C. Anton, A. Toma, L. Cremene, M. Mohorcic, and C. Fortuna, "Power allocation game for interference mitigation in a real-world experimental testbed," in *Communications (ICC), 2014 IEEE International Conference on*, 2014, pp. 1495-1501.

[35] P. Ruckebusch, E. De Poorter, C. Fortuna, and I. Moerman, "GITAR: Generic extension for Internet-of-Things ARchitectures enabling dynamic updates of network and application modules," *Ad Hoc Networks,* 2015.

[36] C. Fortuna and M. Mohorcic, "A Framework for Dynamic Composition of Communication Services," *ACM Transactions on Sensor Networks (TOSN),* vol. 11, p. 32, 2014.

[37] L. D. Mendes and J. J. Rodrigues, "A survey on cross-layer solutions for wireless sensor networks," *Journal of Network and Computer Applications,* vol. 34, pp. 523-534, 2011.

[38] A. Marchiori and Q. Han, "A Two-Stage Bootloader to Support Multi-application Deployment and Switching in Wireless Sensor Networks," in *Computational Science and Engineering, 2009. CSE'09. International Conference on*, 2009, pp. 71-78.

[39] S. Brown and C. J. Sreenan, "Software update recovery for wireless sensor networks," in *Sensor Applications, Experimentation, and Logistics*, ed: Springer, 2010, pp. 107-125.

[40] J. Jeong and D. Culler, "Incremental network programming for wireless sensors," in *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*, 2004, pp. 25-33.

[41] P. Levis and D. Culler, "Maté: A tiny virtual machine for sensor networks," in *ACM Sigplan Notices*, 2002, pp. 85-95.

[42] A. Dunkels, N. Finne, J. Eriksson, and T. Voigt, "Run-time dynamic linking for reprogramming wireless sensor networks," in *Proceedings of the 4th international conference on Embedded networked sensor systems*, 2006, pp. 15-28.

[43] W. Dong, C. Chen, X. Liu, J. Bu, and Y. Liu, "Dynamic linking and loading in networked embedded systems," in *Mobile Adhoc and Sensor Systems, 2009. MASS'09. IEEE 6th International Conference on*, 2009, pp. 554-562.

[44] A. Taherkordi, F. Loiret, R. Rouvoy, and F. Eliassen, "Optimizing sensor network reprogramming via in situ reconfigurable components," *ACM Transactions on Sensor Networks (TOSN),* vol. 9, p. 14, 2013.

[45] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (CoAP)," *Available Online: https://tools.ietf.org/html/draft-ietf-core-observe-06,* 2014.

[46] A. Woo, T. Tong, and D. Culler, "Taming the underlying challenges of reliable multihop routing in sensor networks," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, 2003, pp. 14-27.

[47] N. Baccour, A. Koubaa, L. Mottola, M. A. Zuniga, H. Youssef, C. A. Boano*, et al.*, "Radio link quality estimation in wireless sensor networks: a survey," *ACM Transactions on Sensor Networks (TOSN),* vol. 8, p. 34, 2012.

[48] C. Liu, K. Wu, and T. He, "Sensor localization with ring overlapping based on comparison of received signal strength indicator," in *Mobile Ad-hoc and Sensor Systems, 2004 IEEE International Conference on*, 2004, pp. 516-518.

Bibliography

Publications Related to the Thesis

Conference Paper

A. Bekan, M. Mohorcic, J. Cinkelj, and C. Fortuna, "An architecture for fully reconfigurable plug-and-play wireless sensor network testbed," IEEE Globecom 2015, San Diego, California, USA, December 2015.

M. Celarc, A. Bekan, M. Vucnik, and M. Mohorcic, "Developing API for efficient and secure access to IoT resources and data," in *Proceedings of the 7th Jožef Stefan International Postgraduate School Students' Conference*, 20.-22. 5. 2015, Ljubljana, Slovenia: Jožef Stefan International Postgraduate School, 2015. - ISBN 978-961-92871-9-4. – Book 1, page. 81-91.

J. Cinkelj, M. Sterk, A. Bekan, M. Mohorcic, and C. Fortuna, "Design trade-offs for the wireless management networks of constrained device testbeds," in *Proceedings of the 11th International Symposium on Wireless Communication Systems*, (ISWCS), August 26-29, 2014, Barcelona, Spain. Danvers : IEEE = Institute of Electrical and Electronics Engineers, 2014. - ISBN 978-1-4799-5863-4. - Page. 245-250.

M. Junuzovic, C. Fortuna, A. Bekan, and M. Mohorcic, "Mesh network over the Rime stack using VESNA platforms," in *Proceedings of the 22nd International Electrotechnical and Computer Science Conference ERK 2013*, 16.-18. September 2013, Portorož, Slovenia: IEEE Region 8, Sloveninan section IEEE, 2013. - ISSN 1581-4572. - Book. A, Page. 57-60.

A. Bekan, M. Vucnik, C. Fortuna, and M. Mohorcic, "Sensor as a service using the VESNA sensor platform," in *Proceedings of the 5th Jožef Stefan International Postgraduate School Students' Conference*, 23. 05. 2013, Ljubljana, Slovenia: Jožef Stefan International Postgraduate School, 2013. - ISBN 978-961-92871-5-6. - Page. 108-115.

Other Publications

A. Hrovat, A. Bekan, C. Fortuna, T. Javornik, M. Smolnikar, and M. Mohorcic, "Topologies for connectivity of wireless sensor nodes in a given operational environment," study, IJS delovno porocilo - 11831, 2015. [COBISS.SI-ID 28472359]

C. Fortuna, A. Bekan and M. Mohorcic, "Uporaba metod strojnega učenja za samonastavljiva brezžična zankasta omrežja," study, IJS delovno porocilo - 11909, 2015. [COBISS.SI-ID 28788519]

Biography

Adnan Bekan was born on 29 July 1989 in Foča, Bosnia and Herzegovina. In 2008 he enrolled in the Faculty of Electrical Engineering, University of Tuzla, Bosnia and Herzegovina. Four years later he received his B.Sc. degree in Electrical Engineering. During his study he was working as an external collaborator for several startup companies.

In 2012 he enrolled in the Information and Communication Technologies programme at the Jožef Stefan International Postgraduate School, Ljubljana, Slovenia.

His research interests are in several topics within Wireless Sensor Networks (WSNs) such as interoperability and performance evaluation of communication standards and protocols, IPv6 based WSNs, and integration of different devices and technologies in heterogeneous networks. Within his research work he is mainly focusing on network protocols and architecture of the WSNs, and applications development for the Internet of Things with an emphasis on low-power and lossy networks.

He collaborated on EU funded projects Fed4FIRE (Federation for Future Internet Research and Experimentation) and CREW (Cognitive Radio Experimentation World) and on Slovenian funded project APRICOT (Advanced procedures for interactive composition of sensor networks project), where he was contributing in designing and developing different functionalities of Wireless Sensor Networks (WSN) testbeds.