Building blocks for the internet of things

Building Blocks for the Internet of Things

by Milosh Stolikj

IPA Dissertation Series 2015-19
Printed by Gildeprint, Enschede
Cover by Slobodan Jakjoski

COMMIT/

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).

This work has been supported in part by the Dutch P08 SenSafety project, as part of the COMMIT program.

A catalogue record is available from the Eindhoven University of Technology Library.
ISBN: 978-90-386-3928-4

Building Blocks for the Internet of Things

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een commissie aangewezen door het College voor Promoties in het openbaar te verdedigen op maandag 19 oktober 2015 om 16.00 uur

door

Milosh Stolikj

geboren te Skopje, Macedonië

Dit proefschrift is goedgekeurd door de promotor en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof.dr. J. de Vlieg
1e promotor:	prof.dr. J.J. Lukkien
copromotor:	dr.ir. P.J.L. Cuijpers
leden:	prof.dr.ir. O.J. Boxma
	prof.dr.ing. P.J.M. Havinga (University of Twente)
	dr. L. Almeida (University of Porto)
	prof.dr. H. Corporaal
	prof.dr. M.G.J. van den Brand

PREFACE

This thesis is the result of four inspiring years of work in the System Architecture and Networking Group (SAN) at the Eindhoven University of Technology. It would not have come to life without the help of many people, to whom I would like to express my gratitude.

First of all, I would like to thank my promoter, prof. dr. Johan J. Lukkien, for giving me the chance to pursue a doctoral degree within SAN. I thoroughly enjoyed our collaboration over the years, and I am grateful for giving me freedom to explore my own path, as well as guiding me when needed. My work and research would have not reached this level without your enthusiasm to invest in and to expand my aspirations.

I also want to thank my supervisor, dr. ir. Pieter J. L. Cuijpers, for getting me through the daily hurdles. I appreciate the discussions we had, the useful observations and insights in a lot of strange, new topics, especially since I always had an opponent with a different point of view.

I would like to thank all the members of the reading committee, prof.dr.ir. Onno J. Boxma, prof.dr.ing. Paul J.M. Havinga, dr. Luis Almeida, prof.dr. Henk Corporaal and prof.dr. Mark G.J. van den Brand for reviewing the thesis. I enjoyed reading your comments and discussions, and I believe that your feedback improved the quality of the thesis.

During the past four years, I collaborated with several people. Their contributions were invaluable to me, and shaped my work. I would like to thank Richard Verhoeven for helping out with the technical problems that I struck upon. It is great when you know that there is someone capable of solving practically anything you throw at them. I was fortunate to supervise Nina Buchina as a Master student, and I am grateful for her hard work. I would like to thank everyone involved in the SenSafety project for listening to my ideas, contributing during the plenary meetings, and socializing during the community days.

I especially want to thank Thomas M.M. Meyfroyt, for the effort and energy spent during our collaboration. I really appreciated your strong analytical view on things, and the endless motivation you bring. Thanks to your eagerness and drive, we were able to quickly tackle some interesting problems, and of course, share the experience of presenting them to the rest of the community. This collaboration would not have been possible without the support of prof.dr. Johan J. Lukkien and prof.dr.ir.

Onno J. Boxma. I would like to thank them for their insights during our meetings, and for reviewing our ideas and papers.

I probably would not have done my PhD in the Netherlands if it was not for Jasen Markovski. I would like to thank him and Natasha Jovanovich for reaching out and connecting me with the right people, for advising me throughout these four years, and for helping me settle in. You were an inspiration for me to push further when things were difficult.

I would like to thank my former colleagues from the Institute of Informatics, in Skopje, Macedonia. In particular, I thank Katerina Zdravkova, Nevena Ackovska, Anastas Mishev and Boro Jakimovski, for introducing me to science, and striking that research vibe that I continued to explore further.

My stay in the Netherlands was made a lot easier by my colleagues. I would like to thank Mike Holenderski, Martijn van den Heuvel and Vinh Bui, for making me feel welcome, for helping me with administrative troubles, language barriers, and social endeavours. I would like to thank everyone at SAN, for the shared lunches, coffee breaks, and social events. Thank you for making life at work pleasant and entertaining.

Life in the Netherlands was a lot of fun. For that, I have to thank all my friends in and around Eindhoven, for the many shared nights out, road trips, table tennis sessions, barbecues, etc. Also, I would like to thank my friends from Macedonia, for providing the necessary distractions while being back home, as well as for visiting and helping me explore a new country. Thank you for not forgetting your remote friends and for staying in touch.

I owe a great deal of gratitude to my parents, Elica and Novica, and my sister, Ana, for the constant love and support. Thank you for all your sacrifices, patience and encouragement to become the person that I am now. This thesis is as much your accomplishment as it is mine.

Last but not least, I would like to express deep gratitude to my partner, Aleksandra, for all her love, caring and patience during these four years. Thank you for encouraging me to endure this journey and for being there every step of the way. I hope that I can be as supportive of you as you have been with me.

Milosh Stolikj Eindhoven, September 2015

Contents

1

INTRODUCTION

Electronic devices are getting smaller and more ubiquitous than ever before. We can see electronics in everyday consumer products such as light bulbs, thermostats, and kitchen appliances. With recent technological advances, they are able to connect among themselves and to other devices on the Internet. With the expected massive scaling of such ubiquitous devices, we move towards the Internet of Things (IoT). In this chapter, we give an introduction to the concept of IoT and its major challenges. Then, we list the research topics which are explored in the thesis.

1.1 Introduction

The IoT is a paradigm where every physical object is connected to the Internet and is able to uniquely identify itself to other devices. Such physical objects are referred to as *smart*, meaning that they contain embedded electronics which exhibits some form of intelligence. The term IoT was first used by Kevin Ashton in 1999 [Ash09], to describe a supply chain system using Radio-Frequency Identification (RFID) [Fin03] to a potential customer. Today, we envision the IoT to cover a wide range of applications, such as Smart Grids [ZR13], Smart Cities [She11], Industrial Automation [AIM10], Home Automation [YN13] and Building Automation [JRK12; VD10].

The history of the IoT can be traced in the area of Ubiquitous computing. Mark Weiser proposed the idea of a *smart environment*: "*a physical world that is richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in the everyday objects of our lives, and connected through a continuous network*" [WGB99]. The integration task of this idea is explored in the area of Wireless Sensor Networks (WSNs), where the goal is to build a system of many cheap computational components, called *sensor nodes*, wirelessly connected and jointly working towards a common goal. One instance of a WSN is *Smart Dust*, a large distributed system consisting of thousands of tiny sensor nodes, with sizes as little as one cubic millimetre, using solar energy for power supply, and a communication range of up to several tens of kilometres [Pis97]. This system has been something of a blueprint for WSN research, emphasising the need for systems for low-power operation over inexpensive devices with limited capabilities. WSNs are typically very optimized for a given task, and use special devices, called *gateways*, to connect to other networks, and eventually, end users. This optimization leads to the use of customized protocols, which are incompatible with the existing Internet protocols. In that sense, WSNs can be seen as islands that use gateways to connect to the mainland. The further away the island is, the more complicated the gateway becomes.

The emergence of small implementations of the Internet Protocol (IP) [HC08], specifically built for WSNs, reduced the complexity of gateways. With IP as the common protocol used both in WSNs and in the Internet, it becomes a point of convergence in the IoT. As a result, gateways became much simpler devices, which only have to translate

between different physical media, at the level of common routers. This trend will continue in the future, with the wider deployment of Internet Protocol version 6 (IPv6) [RFC2460] and its adaptation layers for WSNs.

This convergence process supports the main idea behind the IoT, that end-to-end connectivity between any two devices is possible. The devices can be of disproportionate nature, as for instance, a powerful server reading out a temperature sensor, or a user controlling light bulbs via a smart phone. The existence of a common communication infrastructure, using standardized protocols, makes this communication possible. This integration at a large scale is expected to improve many current systems, as transport logistics and various automation systems. However, it will also enable the development of new applications, as smart cities and smart grids (Figure 1.1) [VD10; GBMP13].

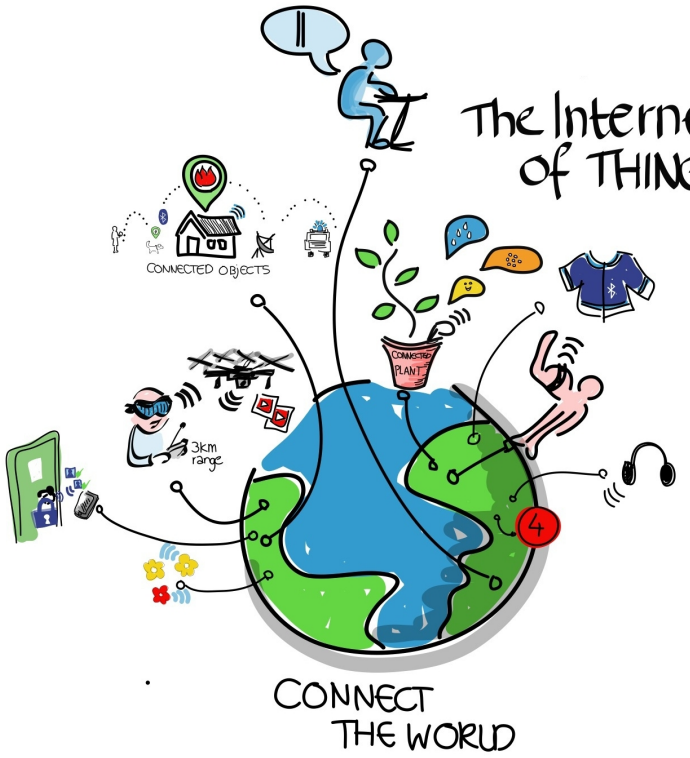


Figure 1.1: The Internet of Things, by Esther Gons, December 2012. Online: `https://www.flickr.com/photos/wilgengebroed/8249565455/`

The IoT is everything but a finished project. In fact, its realization faces many problems. These problems vary from sociological challenges, as making people aware and knowledgeable of the technology, to technological challenges in the system design, data usability, security, and privacy. In this thesis, we address a subset of the latter group of challenges, involving the development of software protocols for the IoT.

1.2 IoT challenges

The technical challenges of the IoT can be identified in several areas.

Connectivity. Connecting trillions of devices in virtually the same network is not an easy task. The heterogeneity of the involved devices makes it even more difficult, since many different physical interconnections can be expected. These differences can completely break certain communications. For instance, city wide ad-hoc wireless networks typically have large latencies, which break timing perspectives of current Internet protocols. IoT solutions need to address this heterogeneity in the design phase.

Power consumption. All electronic devices require power to operate. On the one hand, mains powered devices, as servers and light bulbs, use the power grid. On the other hand, sensors deployed in the wild either rely on batteries, or use some form of energy harvesting. In both cases, due to the scale of the IoT, the devices should be built such that they use as little energy as possible. Furthermore, they open up new challenges, such as the design of sustainable energy harvesting technologies, and the optimization of energy consumption in the local (e.g. data centre, building) or global energy distribution system (e.g. city-wide grid).

System Architecture. The multi-domain nature of the IoT makes it difficult to create a single, killer architecture and application. Simply put, solutions for a certain domain are inapplicable to others, either due to functional requirements, or due to hardware differences. The challenge in the IoT is to construct such architectures, with portability, auto-configuration, integration and connectivity in mind.

Interoperability and integration. The IoT is built by many distinct vendors, using various technologies. Their seamless integration can only be possible if IoT systems are built on top of open standards. There may be multiple standards for the same areas (e.g. different wireless

networking standards), but interoperability between them has to be established (e.g. gateways between different physical networks). We are now at the stage were many of these standards are being defined.

Computational and storage complexity. The devices that comprise the IoT will generate massive amounts of data. These data can be continuous or bursty, and be in structured or unstructured form. In order to extract the most from these data, they have to be transported, stored and analysed. These operations put enormous pressure on networking, storage and computational infrastructure. The challenge in the IoT is to develop and maintain such complex infrastructures.

Security, Trust and Privacy. The penetration of the IoT in daily lives emphasises the need of proper secure solutions. On the one hand, the large number of devices involved makes the design of a completely secure system difficult, as there are many points of potential attack. Then, any solutions have to be portable to a wide set of devices, despite their intrinsic differences. On the other hand, the potentially collectable data and its impact is enormous. Unless proper control is enforced, the IoT can lead to the creation of a dystopian world from Orwell's 1984, which is not what we expect.

From the listed challenges, it is obvious that the development of the IoT depends on the progress of several disciplines, including wireless sensor networks, control systems, cloud computing, and computer security.

1.3 Context and motivation

The motivation for this work comes from the SenSafety project [i]. SenSafety aims to increase safety and security by offering real-time reporting and analysis of potentially dangerous events and providing support to first-responders in such situations (Figure 1.2). SenSafety achieves this goal by gathering and analysing sensory data from any available sensors in a given area. These sensors can be part of a fixed infrastructure, or be completely opportunistic. Two examples of fixed infrastructures are city-wide lighting systems and building automation systems. A city-wide lighting system consists of networks of intelligent light poles, equipped with cameras, sound sensors, presence sensors, air quality sensors, and controllable lights, wirelessly interconnected and capable

[i] http://www.sensafety.nl

of autonomously analysing data. Similarly, building automation systems provide fully autonomous building control, including lighting, air conditioning, heating, and access control, based on connected sensors and actuators inside buildings. Opportunistic sensors come from various end-user devices, including sensors on smart phones and wearable devices. These devices can be used to complement data gathered from the fixed infrastructure, as well as inform the general public of safety issues.

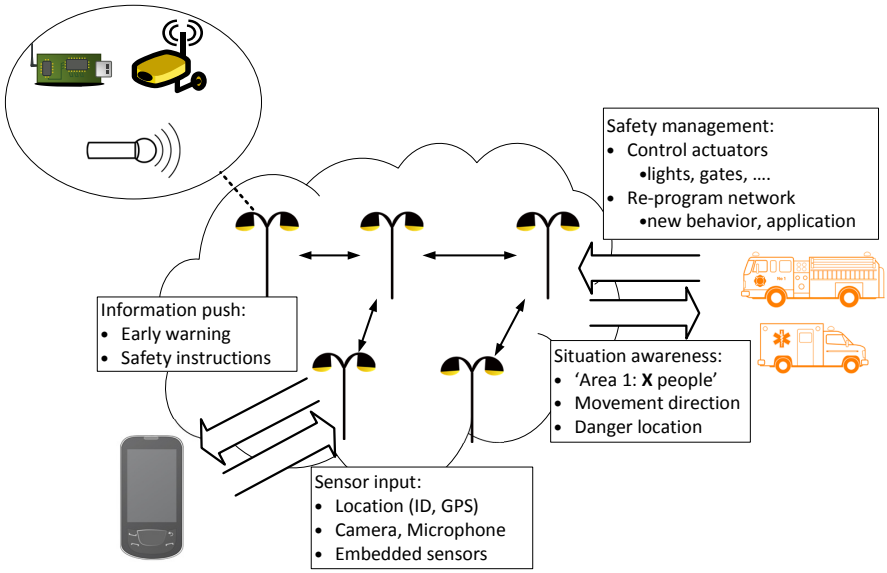


Figure 1.2: SenSafety overview.

Even though many specialized safety systems have been developed, we believe that SenSafety can be treated as another application in the IoT landscape. It closely resembles the Smart City concept, where digital technologies are used to improve common processes in cities, actively engage citizens and enhance their daily lives. In that context, a Smart City provides a platform which can be used to realize the safety applications expected in SenSafety.

The main research focus in SenSafety is in four areas: system architecture, networking protocols, intelligent streaming data sensors, and distributed signal processing. This work addresses the first two areas.

The design of the system architecture is difficult due to the heterogeneity of the devices involved and the combination of different types of sensor input: streaming and event-based. Additionally, the system architecture

should provide support for remotely reprogramming devices for a specific task. Reprogramming is needed for maintenance (e.g. fixing bugs in existing software), or for providing new functionality (e.g. deploying new algorithms for analysing video streams). Reprogramming is also important during the design of intelligent streaming data sensors, which should be able to process audio and video streams in an energy-efficient manner.

Networking protocols in large, wireless networks often have sub-optimal performance due to varying radio conditions, unstable connections, or incomplete network information due to size. The challenge then is how to design protocols which are both energy efficient, as well as robust against varying network conditions.

Finally, distributed signal processing deals with gathering sensor data from different sources, analysing it and making decisions based on current information. This is a challenge due to the unstable networking conditions and the potentially massive amounts of data available.

1.4 Questions and contributions

The focus of this thesis is on the design and implementation of the software architecture for the IoT. In particular, we focus on the integration of resource constrained, low-power devices as part of the IoT infrastructure. We try to answer the following questions:

RQ1: How do we design the software infrastructure for the IoT?

RQ2: How do we combine powerful and resource constrained devices in a single logical network?

RQ3: How do we manage large networks of heterogeneous devices?

RQ4: How do existing IoT specific networking protocols scale in size and in different topologies?

RQ5: What is the impact of low-power wireless radios on different IoT applications?

We address these questions by investigating various aspects of IoT systems. We start by analysing the fundamental requirements and properties expected of IoT use cases. These requirements, such as low energy footprint and portability to constrained devices, pose serious challenges

to existing software protocols. Therefore, we conduct a survey of existing software protocols and discuss their applicability in IoT use cases. From this survey, we select a mix of existing standardized protocols and protocols under development to propose a Service Oriented Architecture (SOA), for realizing IoT applications over low-power wireless networks. SOA allows decomposition of complex systems into applications that use smaller, simpler components with well-defined interfaces [Deu+06]. This decomposition is beneficial for the IoT, as it enables re-use of both hardware and software, and naturally accepts devices with disproportionate capabilities. The proposed architecture is built on open standards, which enables easier interoperability, and is flexible enough to facilitate different IoT use cases, from remote telemetry applications to local control systems.

The successful implementation of the proposed architecture depends on the proper operation of all of the software protocols involved. Due to the scale of the network, and low-power properties of the nodes, this is not a trivial task.

Firstly, the entire software architecture needs to be deployed and maintained. This step covers the distribution of software components to many devices in the network simultaneously. However, remote software update is tedious over low-power wireless networks, as it consumes a lot of energy and can be slow due to the imperfections of the wireless media and the periodic unavailability of nodes. To resolve these issues, we perform a study for reducing the size of software updates. We analyse the trade-off between processing effort and communication cost, and show that in large networks, various data compression schemes can be used even on constrained devices, to greatly reduce energy consumption and improve dissemination time. We apply the same methods to update smart phones as well, with similar results.

Secondly, the service-oriented nature of the architecture requires that a specialized protocol for discovering services is in place. Such a protocol is currently not identified for the IoT. Therefore, we conduct a survey on the applicability of existing service discovery protocols in the IoT domain. The survey shows that Multicast Domain Name System (mDNS) with DNS-Based Service Discovery (DNS-SD) is a potential candidate due to its wide spread usage in Local Area Networks (LANs), strong industry support and flexible deployment options. Upon more detailed analysis, we discover that the performance of the protocol is suboptimal when used in networks where many alternative services are available,

and they are hosted on devices with low-power properties. To resolve these issues, we propose two backward compatible extensions of the protocol. The first extension adds support for proxy server, which cover the imperfections of low-power hosts. The second extension enables clients to discriminate between distinct services as early as possible. These two extensions enable better performance of the protocol in large IoT deployments.

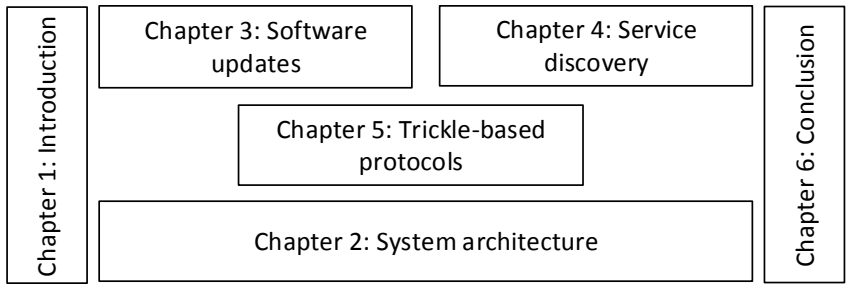


Figure 1.3: Thesis outline.

Finally, for the entire architecture to work properly, appropriate networking protocols for unicast and multicast forwarding need to be in place. The research community is in the process of standardizing two such protocols for the IoT: IPv6 Routing Protocol for LLNs (RPL) [RFC6550] and Multicast Protocol for LLNs (MPL) [HK14]. Both of these protocols use the Trickle algorithm [LPCS04] to quickly propagate information, while limiting redundant traffic. We contribute in the development of these protocols by analysing how they operate over low-power radios, and how robust they are against network topology. We show that in their current design, due to the lack of communication between the network layer and the link layer, MPL can suffer from large delays and significant overhead. We address this problem by modifying the link layer. Then, we demonstrate that the Trickle algorithm is unfair in terms of load distribution, when used in networks of varying density. As a consequence, RPL can suffer from slow route stabilization and redundant traffic. We solve this issue by modifying the Trickle algorithm, to locally optimize fairness properties based on the estimated local network density.

The contributions in this thesis are geared towards a more seamless adoption and implementation of the IoT.

1.5 Outline

The outline of this thesis can be visualized as in Figure 1.3. First, we identify the problems expected in the IoT and propose a possible solution for the system architecture in Chapter 2. Then, we address the problem of updating software in large networks of constrained devices in Chapter 3. Next, in Chapter 4, we cover software protocols for service discovery with emphasis on scalability and interoperability. In Chapter 5 we focus on various issues from the implementation of the Trickle algorithm in routing and multicast forwarding protocols for the IoT. Finally, in Chapter 6, we give conclusions and directions for future work.

2

System architecture

The success of the Internet of Things (IoT) depends on the capability to seamlessly interconnect the plethora of available devices on the market today. Due to the heterogeneity of the market, many architectures and protocols have been developed and are in use. Therefore, in this chapter, we address the system architecture of the IoT with interoperability and scalability in mind. We cover existing state-of-the art protocols which power the IoT, and analyse how they can be used to realize different applications. In the end, we propose a flexible architecture, which is used as motivation for the rest of the thesis.

2.1 Introduction

The multi-faceted nature of the IoT raises significant challenges in the design of its system architecture. The number of possible IoT applications is large, and these applications cover a very broad range of design problems. These applications also come with different requirements, which makes the development of a generic system architecture difficult.

In the IoT, the end-to-end connectivity concept based on the Internet Protocol (IP), mitigates many interoperability-related problems inherited from typical Wireless Sensor Network (WSN) systems. However, end-to-end IP connectivity also introduces additional requirements on all devices that form the IoT, irrespective of their type. At the most basic level, all devices need to run an IP software stack. This is an issue for low-capacity devices with a few kilobytes of memory, as they would have limited space for other applications. Next, the physical properties of certain IoT deployments, as city-wide lighting networks, foresee unstable connections to parts of the network, with variable access latency and possible packet loss. Many existing IP protocols, as the Hyper Text Transfer Protocol (HTTP), cannot handle such environments. To resolve the situation, the community has developed several protocols tailored for the constrained domain of the IoT, ranging from routing, data dissemination to application protocols. The challenge now is to integrate these protocols into one functional unit.

In this chapter, we cover various aspects of system architectures for IoT applications. We start by giving an overview of IoT use cases in Section 2.2. Then, we analyse the requirements for IoT solutions in Section 2.3. Next, we give an overview of the fundamental IoT application design styles in Section 2.4. In Section 2.5, we take a closer look at the main software protocols, which constitute the IoT. In Section 2.6, we present a general system architecture, which can be used to implement different application design styles. Finally, we summarize our results in Section 2.7.

2.2 IoT use cases

The use cases of the IoT can be categorized in three groups: consumer, enterprise and government [Gre14; Cor13] (Figure 2.1).

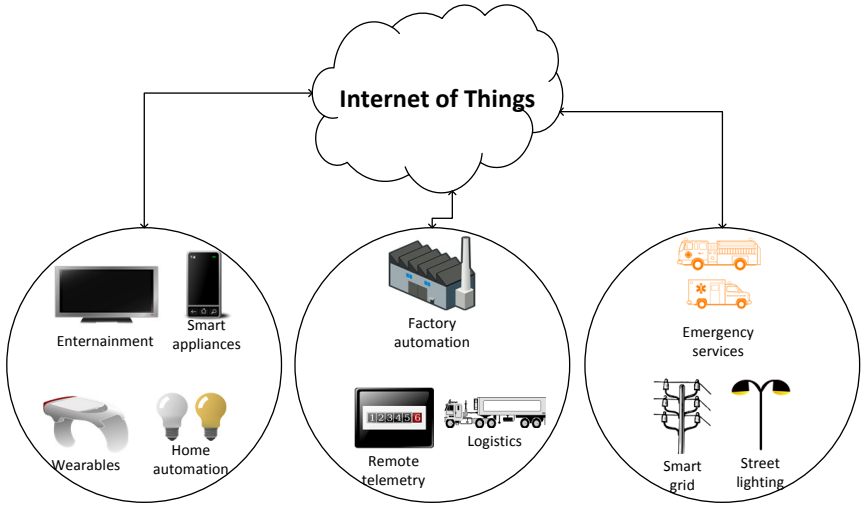


Figure 2.1: List of IoT use cases.

The first category covers various IoT scenarios, which involve a human end-user interfacing with an electronic device. In such use cases, communication traffic is confined between the set of end-user devices and the access point in the IoT. Typical examples of consumer IoT use cases are smart appliances (e.g. smart phones, kitchen appliances [GBMP13]), wearables (e.g. smart watches, fitness trackers, body area networks [Wei14]), home automation (e.g. home lighting [Dan+15], temperature control [YN13]), home entertainment (e.g. smart television sets, connected sound systems [GBMP13]) and other ways which improve daily life. The success of the consumer IoT depends on the ease of use of the technology, plug and play operation, and interoperability between distinct vendors.

The second category covers various industrial automation processes, which can be improved with the expansion of the IoT. Industrial automation relies on fully autonomous operation, where groups of devices work together to achieve a common goal. A typical example is a building automation system [JRK12], where, for instance, the internal temperature of a building is maintained by close cooperation between temperature sensors, ventilation and heating elements, shade controllers etc. These systems often use external servers for reporting and storing periodic information about the latest state of the system. Therefore, industrial automation systems need support for peer-to-peer communication between nodes in the network, proper identification of devices based on

their properties, reliable operation without human intervention, and other similar features.

The third category of IoT use cases is made possible by integrating various IoT systems into a larger system. For instance, the energy usage from every household, which is already available via home automation systems, can be used to optimize the global energy distribution grid [ZR13], and waste collection systems [She11]. Similarly, telemetry from cars, available from automotive systems, can be used to predict and to avoid traffic congestions, detect accidents, better inform governmental institutions, and optimize performance of street lighting systems [GBMP13; AIM10]. Such global applications can be possible if the sub-systems are built in an open fashion, with interoperability and data re-use in mind.

2.3 Requirements

The requirements of IoT systems come from various sources. Firstly, the IoT domain defines its own requirements for the system design and system behaviour. Secondly, the hardware devices which form the IoT come with certain requirements and constraints. Thirdly, the deployment characteristics of various IoT applications influence the overall system design. Finally, network properties are an important factor in the selection of appropriate software protocols. We analyse each of these four factors and provide an overall list of requirements in the following subsections.

2.3.1 Domain constraints

The IoT is built on the premise that the existing Internet will grow massively, to include the previously described smart objects, and that end-to-end IP connectivity will be possible between any endpoints in the network. Therefore, any solutions for the IoT need to be scalable to operate in large networks, and need to integrate all devices.

The proper operation of such an infrastructure depends on the interoperability of solutions for various sub-systems, possibly originating from various vendors. This interoperability can only be possible if the core protocols are built on commonly accepted open standards. This view has been backed by the community, with calls for "commitment to

openness from companies" [Ger14], "curated openness - standardization of a few core functions..." [MM12], and acceptance and use of common standards [Sub14; Yoo15].

2.3.2 Hardware constraints

The massive scaling in the number of connected devices in the IoT is challenging both from an economical and an infrastructure point. Firstly, since these devices have to be purchased by the interested parties, they have to be relatively low in price. However, cheaper devices are always inferior to their more expensive counterparts. Secondly, these devices have to be powered by either the current energy grid, or be adapted to use energy harvesting methods. Finally, in long-term deployments, which often involve placing devices at hardly accessible places, as smart parking systems or forest fire monitoring systems, devices are expected to operate for long periods of time on battery power, without human intervention. In any case, it is preferable that they are able to operate using as little power as possible.

Due to these economical and physical constraints, many of the devices that comprise the IoT exhibit certain limitations. These limitations can be seen as lack of features which are taken for granted in today's common Internet-enabled devices. Typical examples of such limited features are small memory, lack of constant power supply, and slow processor speed. In the rest of this text, we refer to such devices as *constrained* nodes.

We categorize constrained nodes into different hardware classes based on the Internet Engineering Task Force (IETF) informational document [RFC7228]. The categorization is made on two distinguishing features: the processing capabilities of current devices on the market (Table 2.1, henceforth *class* or j) and the expected power consumption in various deployments (Table 2.2, henceforth *power profile*, covered in the next section) [i]. The categories shown in Table 2.2 will inevitably shift over time, with new, more powerful devices appearing on the market. However, due to the cost factor involved and the energy restrictions shown in Table 2.2, the improvements in the constrained sector will most likely move at a slower pace, compared to the high-end devices.

[i] Note: In the remainder of this thesis, we use the following units: b = bit, B = byte, 1 KiB = 1024 B, 1 MiB = 1024 KiB, 1 kB = 1000 B, 1 MB = 1000 kB.

With the exception of class C0, which only corresponds to power profiles P0 and P1, all other constrained node classes C1-C9 can be required to operate with any given power profile P1-P9. The class determines the functional capabilities of the constrained nodes: nodes of class C1 and C2 can host a real time operating system with a limited IP stack, and can control basic peripherals as sensors and actuators, but cannot support additional functional code. Nodes of class C3 are an upgrade, and besides basic functionality, have enough capacity to run additional supportive code. Nodes of class C9 are the very high-end of resource constrained devices, and correspond to powerful portable devices.

Table 2.1: Processing class: constrained classes of nodes, based on available computing resources.

Class	Non-volatile memory (ROM)	Volatile memory (RAM)	Platform	CPU freq.	Examples
C0	$\leq$ 1KiB	0B	8 bit	0MHz	Passive RFID tags
C1	$\leq$ 64KiB	$\leq$ 2KiB	8 bit	$\leq$ 16MHz	ATmega644RFR2
C2	$\leq$ 128KiB	$\leq$ 10KiB	16 bit	$\leq$ 16MHz	RC230X, TI MSP430
C3	$\leq$ 256KiB	$\leq$ 128KiB	32 bit	$\leq$ 96MHz	MC13224, STM32W
C9	$\geq$ 512MiB	$\geq$ 128MiB	32, 64 bit	$\geq$ 400MHz	Raspberry Pi

2.3.3 Power constraints

The power profile limits the complexity of the code which can be run on given nodes, as well as the node's availability in the network. Nodes with power profile P0 are activated by an external source, and are able to deliver information only to the source. Therefore, they are not accessible to other nodes in the network. Nodes with power profile P1 are available in the network during short bursts, during which they can be used to quickly deliver or fetch data. They may be treated as P0 nodes, or as nodes with extremely large offline periods. Nodes with power profile P2 need to efficiently manage their power consumption, so often employ some form of Radio Duty Cycling (RDC). As a result, they can be seen as periodically online nodes, but with large latencies for access, which can render existing IP protocols as inapplicable. Nodes with power profile P3 exhibit some form of a power constraint: they may be part of a large network, with overall energy restrictions, or may

have interleaved periods of battery supply and mains charging. A typical example are nodes in street light poles, which are connected to the grid during the night, but run on batteries during the day. These nodes can switch between power saving mechanisms, and can exhibit low-power behaviour, but with moderate access latencies. Lastly, nodes with power profile P9 do not have power constraints that influence their operation within the IoT.

Table 2.2: Power profile: constrained classes of nodes, based on power requirements.

Profile	Type	Examples
P0	passive	RFID tags
P1	energy harvesting	buttons, switches
P2	ultra low-power	battery operated, inaccessible devices
P3	low-power	large scale deployments
P9	mains powered	light bulbs, infrastructure devices

2.3.4 Network constraints

The presence of constrained nodes in a network has an impact on the network operation. The IETF defines networks formed of a large portion of constrained nodes as constrained-node networks. A typical example of a constrained-node network is a Low-Power and Lossy Network (LLN), where constraints come in the form of limited memory and processing capabilities, various interconnection links and limited power supply of constrained nodes. Due to these constraints and radio medium properties, LLNs face considerable data loss at the physical layer, asymmetric links, temporal unavailability of nodes, low throughput, high latency, and other similar properties. As a result, LLNs have difficulty attaining network characteristics which are taken for granted with link layers commonly used in the Internet.

2.3.5 Summary: Issues and Requirements

Based on the given constraints, we summarize the key properties and issues that system solutions and software protocols for the IoT should address. These broad properties can be made exact when applied to a specific use case and application:

R1: Scalability. Scalability is an important issue, since the number of nodes in the system can grow drastically (e.g. thousands of devices in a single network), which results in growth of data produced by these nodes, and an increase in the number of possible applications. Therefore, scalability has to be addressed on all levels of the system architecture. For instance, scalability in size in routing and forwarding protocols refers to the capability to add new nodes in the system, without overloading the capacity of the existing nodes. This can be achieved by applying de-centralized design, limiting control traffic, condensing topology information. At the application level, among other things, scalability poses a challenge in the identification of devices and resources, their discovery and access in a timely manner. Finally, for security solutions, scalability involves distribution and management of encryption keys, distributed authorization and attestation.

R2: Connectivity. Since IoT systems can partially consist of LLNs, they need to operate in constrained environments, such as networks which exhibit:

R2a: Small data payload, as for instance, in IEEE 802.15.4 [IEEE15.4e], where the maximum usable layer 2 payload is around 80 bytes, or the 22 byte maximum of Bluetooth Low Energy [Blu12].

R2b: Low bandwidth (e.g. 250 kb/s in IEEE 802.15.4, up to 1 Mb/s in Bluetooth Low Energy).

R2c: Large latency for access, due to multi-hop traffic forwarding in LLNs, or due to using radio duty cycling.

R2d: Temporal unavailability of parts of the network.

R3: Self-awareness and responsiveness. The distributed nature of the IoT implicitly requires systems to be able to automatically organize themselves within a short period, to fulfil the given tasks. This self-organization includes the initial setup of the infrastructure, such as setting up routing paths, clustering of nodes, as well as setup and maintenance of applications (i.e. application deployment and subsequent updates), automatic discovery of devices, resources and services needed for the application, resilience to failing nodes and so forth. Specific metrics for each of these functionalities

can be derived from the particular use cases. Due to the long-term nature of particular IoT use cases, such systems they should be capable to be expanded at runtime, with new hardware and software components.

R4: Interoperability. The IoT is built by integrating hardware and software components and systems, built by multiple vendors, using various technologies. This integration can be made possible only if common open standards are in place. Interoperability at different levels can be verified through certification authorities, as for instance, the *WiFi Certified* trademark by the WiFi Alliance [ii], and the *Bluetooth* trademark by the Bluetooth Special Interest Group [iii], or through integration tests, as the European Telecommunications Standards Institute (ETSI) Plugtests [iv].

R5: Portability and heterogeneity. Depending on the domain, heterogeneity can come in hardware (i.e. various devices, communication interfaces, architectures) and software (i.e. various software stacks, protocols, policies). Due to this heterogeneity, IoT systems, and many of the software components that comprise it, should also be portable to constrained nodes, with small memory footprint and low computational capabilities, as listed in Table 2.1.

R6: Security and privacy. Security is essential in most IoT use cases, as they can either be of critical nature, as for instance, infrastructure monitoring, or can carry sensitive data, as in health systems. Security issues include data integrity, device and user authentication, and encryption of communication links. Therefore, security should also be addressed on all levels of the system architecture.

2.4 IoT application design

A crucial step in the design of the IoT infrastructure is the decision where the application logic should be placed. The two prevailing extreme views are the distributed and the cloud-based approach.

In the **distributed computing paradigm** (Figure 2.2), the application logic remains within the LLN, and is distributed among nodes in the

[ii] http://www.wi-fi.org/
[iii] https://www.bluetooth.org/
[iv] http://www.etsi.org/about/what-we-do/plugtests/

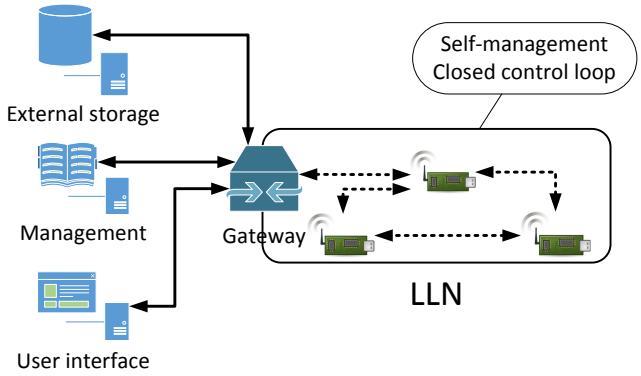


Figure 2.2: Distributed IoT infrastructure. Application logic is mostly confined in the LLN domain, with limited need of outside interaction.

network. This approach keeps communication and data local, inside of the LLN, which is important for both performance and safety. Additionally, hardware cost is decreased, as there is no need of additional expensive hardware. Examples of fully distributed IoT systems are SOFIA [TPSBO09], COSMOS [BMT03], and SensibleThings [FKOJ14].

The main drawback of this paradigm is that due to the limited processing capabilities, the application logic has to be simple enough to run on constrained nodes. Therefore, most applications are in the form of closed control loops between sensors and actuators. Implementation-wise, the direct interaction between nodes (point-to-point traffic) poses a challenge in large networks. Namely, the infrastructure would have to either maintain routes for each control loop, or resort to using only group communication patterns (i.e. broadcast or multicast). Both solutions are non-trivial and can be detrimental to network performance.

In the **cloud computing paradigm** (Figure 2.3), smart objects are seen as information providers. Sensed data is carried via an LLN to the cloud, in the form of multipoint-to-point communication. Then, all complex data processing and decision making take place in the cloud. When the decision involves actuation, it is fed back into the LLN in the form of a point-to-multipoint command. The cloud computing paradigm takes away most of the complexity from constrained nodes, which remain responsible only for sensing/actuation, simple data aggregation and forwarding. However, this comes at the cost of relatively high latency for detecting events. The communication patterns in the cloud computing paradigm are suitable to be implemented over a tree-based organiza-

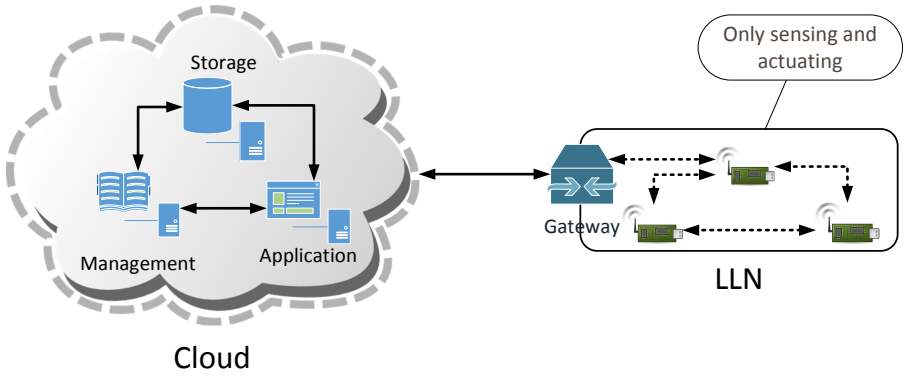


Figure 2.3: Cloud-based IoT infrastructure. Devices in the LLN are used only as information providers, with application logic taking place in the cloud.

tion of nodes, with gateways acting as the root. Then, all traffic is either directed upwards, towards the root, or disseminated downwards to the entire tree. Typical examples of cloud-based IoT systems are WoTKit [BL12], SicsthSense [McN+14], Xively [Inc15].

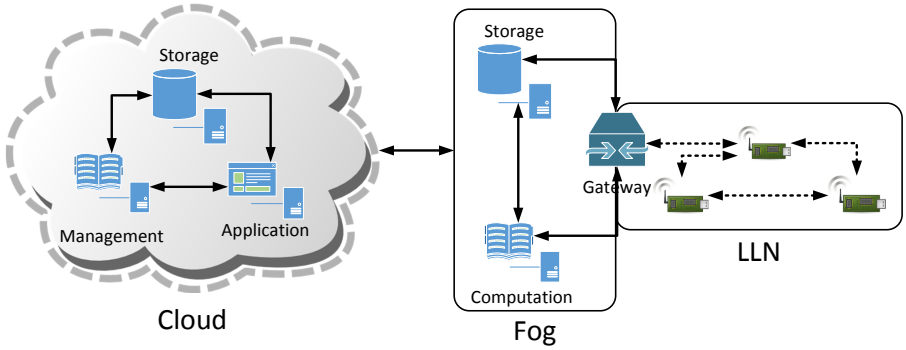


Figure 2.4: Fog-based IoT infrastructure. Application logic is located in the 'fog', close to the boundaries of the LLN.

Fog computing (Figure 2.4) is aimed to improve the performance of cloud computing by moving part of the cloud capabilities locally, at the edge of the network [BMNZ14]. It is an intermediate step between the distributed and the cloud based approach in the IoT, with additional processing/storage capacity located on the borders of LLNs. This proximity can improve the overall computational and storage capacity of the LLN, with smaller latency compared to the cloud based approach. Typical examples of fog-based IoT systems are Mobile Fog [Hon+13], BETaaS [MTVDG13] and SmartGateway [AH14].

2.5 Software protocols for IoT

Currently used IP based protocols in the Internet are unusable in the IoT due to the constraints of LLNs. As a result, several new protocols aimed at LLNs have been developed and standardized. We now present an overview of the most important ones, according to the software layer where they operate. The protocols are summarized according to the 4-layer internet model [RFC1122] in Figure 2.5.

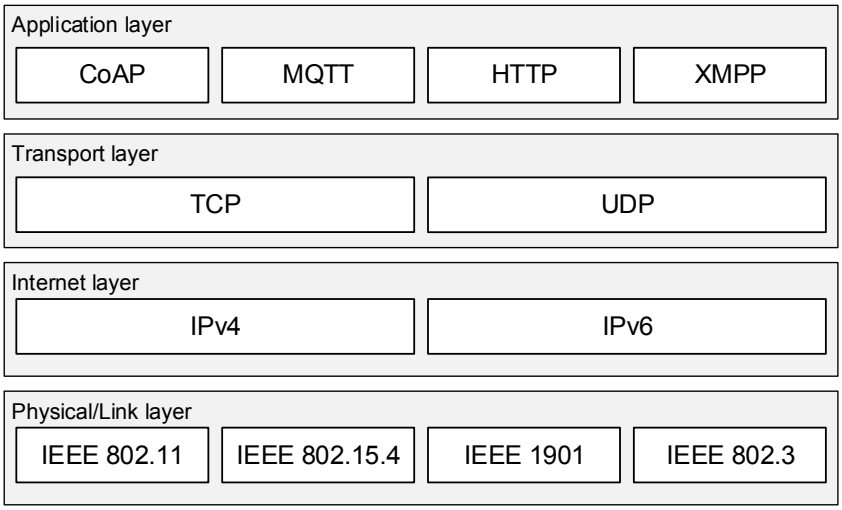


Figure 2.5: List of various IoT protocols in the 4-layer Internet model.

2.5.1 Physical and link layer

Several technologies have been seen as potential solutions for different aspects of the IoT at the physical and link layer. They range from semi-long range wireless solutions, as the IEEE 802.15.4 standard for low-rate wireless personal area networks [IEEE15.4e] and the Low-Power WiFi standard (IEEE 802.11) [KLKG15]; short-range wireless connectivity, with Bluetooth [Blu12], Radio-Frequency Identification (RFID) [Fin03] and Near-field Communication (NFC) [Wan11] being most dominant; and wired technologies, as IEEE 1901 Power Line Communication (PLC) [FGHHV01] and Ethernet [IEEE3bt]. Due to the vari-

ous deployment requirements, all technologies are expected to play a role in the IoT, and interconnections between them are required.

The focus of this thesis is on LLNs built on top of the IEEE 802.15.4 standard, called a Low-power Wireless Personal Area Network (LoWPAN). LoWPANs are characterized with small packet sizes of at most 127 bytes at the physical layer, low bandwidth ranging between 20 kb/s, 40 kb/s and 250 kb/s, depending of the radio frequency used (868 MHz, 915 MHz, or 2.4 GHz, respectively). LoWPANs can operate in an ad-hoc manner, and can span up to thousands of nodes. Even though the application layer is agnostic of the physical layer and the link layer, these two layers are of particular importance in the design of the network layer, as we will show in Chapter 5.

2.5.2 Network and transport layer

The network and transport layer are a convergence point in the Internet, and potentially, in the IoT. In modern Internet applications, end-to-end IP connectivity is available between any two endpoints, with the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) as prevalent transport layer protocols. Any Internet-enabled device is capable of processing IP packets, irrespective of the physical media through which they are transferred. Then, a variety of application protocols can use this information for different purposes (Figure 2.6).

End-to-end IP connectivity in LoWPANs is one of the open research areas in the IoT. As a solution, the IPv6 over LoWPAN (6LoWPAN) adaptation layer is proposed [RFC4919], which provides an adaptation layer for delivering Internet Protocol version 6 (IPv6) packets to and from LoWPANs. 6LoWPAN resolves many challenges posed by the incompatibility between IPv6 and IEEE 802.15.4, among which:

- Support for packet fragmentation and packet reassembly [RFC4944]. IPv6 defines a maximum transmission unit of 1280 Bytes, while IEEE 802.15.4 is limited to a maximum of of 127 bytes at the physical layer, of which, depending on the encryption mechanism used, between and 81 and 102 bytes are useful for payload. Therefore 6LoWPAN defines how large IPv6 packets can be fragmented for delivery over IEEE 802.15.4.

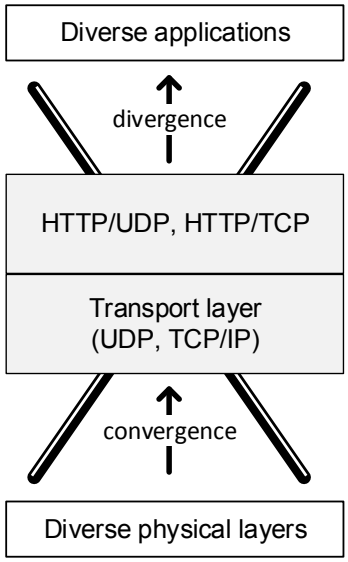


Figure 2.6: The Internet/Transport layer is point of convergence in the current Internet architecture. Courtesy of Johan J. Lukkien. (Lecture notes on Internet of Things. [Online] `http://www.win.tue.nl/~johanl/educ/IoT-Course/IoT-01-v2%20The%20Things.pdf`).

- Support for header compression [RFC4944; RFC6282]. IPv6 packets have a default header of 40 bytes, which leaves only 40 bytes for payload when used over IEEE 802.15.4. Therefore, 6LoWPAN defines mechanisms for compression of the IPv6 header, to reduce overhead. Some of the mechanism can also compress transport layer headers as well.
- Address auto-configuration [RFC4944]. IPv6 uses hierarchical 128 bit IPv6 addresses, while IEEE 802.15.4 uses extended 64 bit addresses, or 16 bit short addresses within LoWPANs, and maintains a separate 16 bit LoWPAN identifier. 6LoWPAN defines a stateless address auto-configuration algorithm for generating a pseudo 48 bit address from the 16 bit short address and the LoWPAN identifier.
- Routing support. This includes mesh routing, within LoWPANs, which is transparent to nodes outside of the LoWPAN, as well as routing between the IPv6 domain and the LoWPANs. For routing within LoWPANs, a graph-based overlay routing protocol, called

IPv6 Routing Protocol for LLNs (RPL) [RFC6550], is standardized, and a stateless multicast forwarding protocol called Multicast Protocol for LLNs (MPL) [HK14] is under development.

2.5.3 Application layer

Traditionally, application layer protocols for WSNs have been particularly tailored for the overall application area of the WSN. This approach has been abandoned in the IoT, where application protocols are developed to be as general as possible. Currently, the most popular application layer protocols in the IoT community are RESTful HTTP, Constrained Application Protocol (CoAP) [SHB13], the Message Queuing Telemetry Transport for Sensor Networks (MQTT-SN) [SCT09] and the Extensible Messaging and Presence Protocol (XMPP) [RFC6120; KK12a; Ben+13]. The data payload of these protocols is usually encoded according to the Extensible Markup Language (XML) [Bra+08], the JavaScript Object Notation (JSON) [RFC7159] or the Efficient XML Interchange (EXI) [SKPK14] format.

Table 2.3: Comparison of application protocols for the IoT.

Protocol	Communication	Transport	Scalability	Security
RESTful HTTP	Client/server	TCP	Limited	HTTPS
CoAP	Client/server Publish/subscribe	UDP	Excellent	DTLS
MQTT-SN	Publish/subscribe	TCP/UDP	Excellent	TLS/SSL
XMPP	Client/server Publish/subscribe	TCP	Fair	TLS/SSL

RESTful HTTP is one of the leading protocols in the so called Web of Things [GT09], a variant of the IoT, where the RESTful approach of application development is extended to include 'things' or smart objects. RESTful HTTP is a realization of the Representational State Transfer (REST) architectural style over HTTP. REST is a style for developing distributed applications by distributing application logic in components, which can communicate using a stateless, cacheable client-server communication protocol. This allows separation of concerns, such as servers dealing with data access, and clients only handling user interface. To improve scalability, components can be layered hierarchically. RESTful

HTTP uses HTTP as the unifying communication protocol, which defines the mechanism for accessing components. Due to the overhead of HTTP, RESTful HTTP within the IoT is mostly used in small-scale consumer networks of more powerful devices, such as home entertainment systems.

CoAP is an improvement of RESTful HTTP, in the form of a generic application protocol for realizing RESTful architectures in constrained networks. It is optimized for Machine to Machine (M2M) applications, with emphasis on simplicity, low memory footprint, small message overhead, operation over unreliable links and asynchronous message exchanges. Furthermore, unlike HTTP, CoAP supports multicast operation for one-to-many traffic patterns, and resource discovery for enabling automatic configuration and deployment. For interoperability with the Web of Things, CoAP defines a stateless HTTP mapping, which allows CoAP resources to be accessed by RESTful HTTP clients. CoAP has been proposed for several IoT applications, including transport logistics [KBBG11], building control [SL12], and smart metering [ASGT12].

MQTT-SN is a lightweight publish/subscribe communication protocol aimed at M2M applications between constrained nodes over lossy, low-bandwidth links. It is a many-to-many communication protocol, where clients communicate to other clients via a central broker. Each client can post messages to the broker, categorized under an arbitrary topic. Then, other clients can choose to subscribe to an arbitrary number of topics. Whenever a topic is updated, all subscribed clients will get notified. The broker itself does the matching between clients, and is agnostic to the message content. MQTT is primarily used in remote telemetry scenarios, where periodical sensor readings or events are reported to an external party, such as home energy monitoring and remote patient monitoring [L+12].

XMPP is an XML based protocol for message delivery between endpoints. It was initially designed for real-time data streaming and Instant Messaging applications, using XML schemas for message transport over TCP links. It supports both publish/subscribe and client/server operation. Since recently, a set of IoT related extensions are under development [Wah14a; Wah15a; Wah15b; Wah15c; Wah14b; WK14], including extensions for reducing XML complexity, for reading and controlling sensor and actuators. XMPP has been proposed for usage in remote patient monitoring [Baz+12], disaster management [KGKS11], and smart homes [WKKK13].

To summarize, there are several application protocols which may be used in the IoT. Each of them has its benefits and drawbacks, and the selection of the most appropriate one depends on the specific use case.

2.6 IoT system architecture

Due to the broad scope of the IoT, it is difficult to predict whether a single architectural style or a single application protocol will prevail. The more likely outcome will be that they will all be used, for realizing different applications. Therefore, we focus on defining a general architecture, which can support the mentioned application protocols, to realize the described application paradigms. The goal is to build the IoT as an open platform, which is agnostic to a particular domain or a particular application. This view has been backed by recent research trends, as the Architectural Reference Model (ARM) by the Internet of Things - Architecture (IOT-A) consortium [TA13], and the M2M Functional Architecture by ETSI [Eur11].

As an architectural style, we take a hybrid approach between fog and completely distributed computing. We assume that an LLN is built of a mixture of extremely resource constrained (classes C0 to C2) nodes, and some more powerful nodes (classes C3 and above). The former group of nodes can only deliver basic functionality, which is implemented at production or commissioning time. These devices are not expected to support software changes other than parameter changes and upgrades/bug fixing scenarios. The latter group of nodes is more flexible, and can deliver additional functionality to the network. These nodes can be re-programmed at run time, and support parameter changes, application maintenance (e.g. addition/removal of components) and system updates.

Next, we adopt the Service Oriented Architecture (SOA) design pattern, which enables separation between *services* and *applications*. A service is a self-contained unit of functionality, whose internal behaviour is abstracted to clients. An application is then built by connecting services, for providing a higher-level functionality. This allows for late binding between applications and services, and ability for incremental growth. Underneath the service layer lies a standardized platform, or so called Operational Service Layer, which provides utility methods to services. This view is a refinement of similar service-based middleware solutions

for the IoT solutions, which aim to abstract physical objects and their communication capabilities as services [Spi+09; AIM10].

This hybrid approach and the adoption of SOA gives enough flexibility to implement any of the three previously described paradigms, and can be used with any of the described application level protocols. Due to the de-coupling of application, services and basic functionality provided by the platform, system designers can migrate between the different paradigms by moving the services between the LLN, the fog and the cloud.

When compared to other architecture efforts, this architecture lies between the ARM efforts of IOT-A and the M2M Functional Architecture by ETSI. On the one hand, ARM aims to model the entire IoT domain, the entities which contribute to it and their relationships. ARM consists of a Reference Model (RMO), a Reference Architecture (RAR) and a set of guidelines. RMO provides a high-level description of the IoT domain for which the system architecture is built. This description includes a domain model, which gives a general specification of the domain, an information model, which explains how IoT knowledge is to be modelled, and a communication model, which specifies the communication patterns. RAR provides different views and perspectives of the architecture, according to the stakeholders of the system. Finally, by using the guidelines, the RMO and RAR can be implemented to build a concrete architecture. As such, ARM has a very broad scope, and needs further implementation before an explicit architecture is built.

On the other hand, due to its industry backing, the Functional Architecture by ETSI is very specific, and focuses on defining a service-abstraction layer, which serves as a middleware between applications and devices. Applications provide the wanted behaviour of the network, by accessing and using services using open interfaces. The architecture focuses exclusively on M2M communication, and integrates both legacy devices, which have no service capabilities, as well as constrained devices, using gateway applications. The entire system is resource based, and follows the RESTful approach. The exact description of resources, and their mappings to HTTP and CoAP is further specified in [Eur14]. Our architecture is compatible with the view taken by ETSI. While we take a more general view in the definition of the concepts, with the definition of a *platform* for common services, the final implementation of both architectures can be the same.

2.6.1 Concepts

Next, we describe the fundamental concepts of the proposed architecture, including the services, the applications and the platform.

2.6.1.1 Service

In SOA, services are the basic building blocks which can be used to build an application. According to the Organization for the Advancement of Structured Information Standards (OASIS), a service "is the mechanism by which needs and capabilities are brought together" [Mac+06]. For the IoT domain, we can further constrain this definition, by limiting the entities which provide services, so called Service Providers (SPs), and the entities which use services, so called Service Clients (SCs), to be only *software components*. The information necessary to interact with a service, as the protocol used, interfaces for network access, inputs, outputs, pre-conditions, post-conditions and general semantics, are promoted through *service descriptions*, associated with each service. Services can be local, as for example, a service providing the reading of a sensor connected to the same node, or can be composite, as for example, an aggregation service, which collects and processes data from other services in the network. The life cycle of a service can be defined with the following steps:

S1: Specification, design and implementation of a software component capable of delivering a given functionality to a network.

S2: Analysis and testing, resulting in a profile of the software component, with information on resource usage and performance metrics.

S3: Deployment, i.e. the placement of the component onto a physical platform that can execute it.

S4: Activation, i.e. initial start-up of the component which makes the service available to the network.

S5: Execution of the service.

S6: Termination of the service, which includes de-activation and removal of the software component.

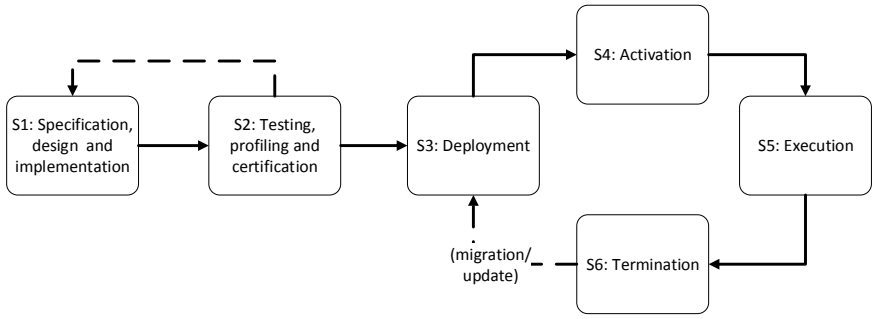


Figure 2.7: Lifecycle of a service.

From the given life cycle, shown in Figure 2.7, steps *S1* and *S2* are done offline, while steps *S4* to *S6* are done at runtime. Step *S3* can be done at manufacturing or commissioning time for all services, or even at runtime, for services hosted on powerful nodes. The underlying assumption for the given specification is that there is a platform capable of running software components.

2.6.1.2 Application

In the context of the IoT, we can define an application as the emergent behaviour of a system. In SOA, we implement an application as an assembly of software components, services and their interactions which result in the wanted behaviour. In other words, an application is built by using available services in a network, connecting them and providing logical flows based on their output. The application specific logic, the coordination of services, additional processing of input from services is located in software components. Each application goes through the following life cycle (Figure 2.8):

A1: Specification, design and implementation, resulting in one or more software components.

A2: Deployment of the software components on a single node or on a set of nodes.

A3: Execution of the software components. At run time, the application is able to:

a) Discover available services.

b) Access services.

c) Connect services, i.e. instruct one service to use another service as input.

d) Process and react to input from services.

A4: Termination of the application, including de-activation and removal of the software components.

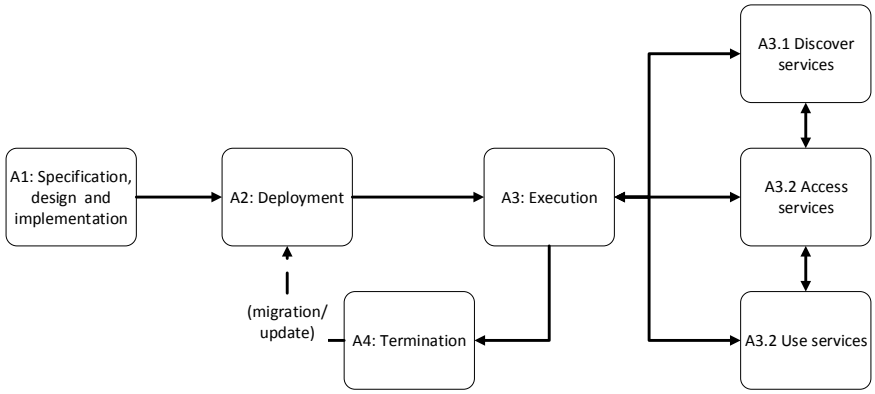


Figure 2.8: Lifecycle of an application.

2.6.1.3 Platform

The role of the platform is to provide common primitive functionality needed by most services. It is a distributed middleware, which lies on top of the operating system on each node, and contains the necessary features to enable service orchestration in the network. As such, the platform hides the inner working of the lower layers and abstracts from their complexity. In a certain way, it can be seen as the operating system for networked services. Unlike other architectures, as [AIM10; KLD12], which provide service abstractions to hardware-specific features (e.g. sensors and actuators), here the platform is open-ended, and can be used to host and provide arbitrary services as well.

For the IoT, we want the platform to at least provide the following functionalities:

P1: Manage services in a network (Install, Start, Stop, and Uninstall software components on an individual node or on a set of nodes).

P2: Expose a service to the network.

P3: Discover an interface for accessing a service in the network based on selection criteria.

P4: Access a service using a known interface.

P5: Enforce security restrictions.

In the current IoT world, application protocols are built to provide *P1-P5*, which poses integration problems as complex application-level gateways are required to provide interoperability with other application protocols. Therefore, we leave *P4* and *P5* to the application protocol, and we take *P2* and *P3* in a separate service discovery layer, which expose constructs for describing, publishing and discovery of service in a network. *P1* then uses both layers to deliver the wanted functionality: the service discovery layer is used to find nodes which can host arbitrary software components, and the components themselves are transferred via the application protocol. Finally, the security component maintains proper protection and reliability of the platform.

2.6.2 Viewpoints

For describing the proposed architecture in more detail, we resort to using several architectural views on the system, concerning different stakeholders. Rozanski et al [RW11] define a view as "*a representation of one or more structural aspects of an architecture that illustrates how the architecture addresses one or more concerns held by one or more of its stakeholders*". The generic creation of a view is done via the concept of a *viewpoint*, defined according to the IEEE Standard 1471 [IEEE1471] as "*a collection of patterns, templates, and conventions for constructing one type of view. It defines the stakeholders whose concerns are reflected in the viewpoint and the guidelines, principles, and template models for constructing its views*". We now show the following common architectural views: Context, Functional, Development and Deployment/Operational.

2.6.2.1 Context view

The context view describes the main elements of a system, and their interactions with the environment. These are shown in Figure 2.9.

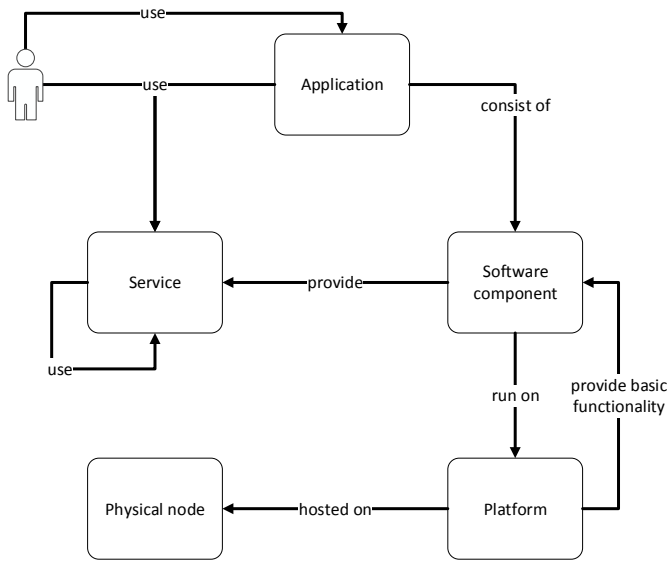


Figure 2.9: Context view of the architecture.

End-users interface with the system through applications, or directly, by accessing services. Applications are built of software components, which provide services. Software components are executed in the platform, which is located on a physical devices. Finally, each platform contains a set of basic software components, which expose basic services to applications and users.

2.6.2.2 Functional view

The functional view, shown in Figure 2.10, describes the main elements of the system, which are responsible for delivering its functionality. It is a three-layer model, where applications are built on top of services, which use the platform. The platform itself consists of services, which are made available for use to other services, both located on the same physical device, and on the network. The critical functional components of the platform here are:

- Networking component, for enabling connectivity (e.g. IP networking stack).

- Access component, the application protocol for accessing services (e.g. CoAP, MQTT-SN).
- Management component, for controlling services and monitoring the system.
- Discovery component, for publishing service descriptions and discovering interfaces for accessing other services.
- Security component, for encrypting traffic and authorization.

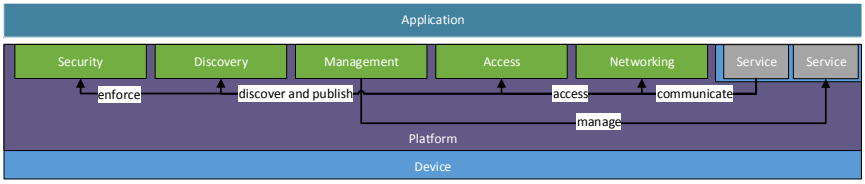


Figure 2.10: Layered model which shows the functional elements of the architecture and their relationships.

2.6.2.3 Development view

The specification of applications depends on the abstraction level provided by the underlying platform, as well as the available tooling. In service oriented WSNs, this typically varies between development of software components in:

1. Low-level programming language (e.g. C code for real time operating systems as Contiki [DGV04], TinyOS [Lev+05]);
2. Interpreted or scripting language (e.g. Open Service Architecture for Sensors (OSAS) [BLV09], Mate [LC02]);
3. High-level domain-specific language (DSL) which is compiled in native code before deployment (e.g. SEAL [EJS13], Midgar [GG-BECF14]).

In this view, illustrated in Figure 2.11, we assume that a DSL is available, which provides abstractions for services. The source code written by developers, is first compiled into a low level source code for an embedded operating system. In this step, service abstractions are expanded into source code constructs available in the embedded operating system. Then, the produced source code is compiled in a native executable, with

appropriate features taken from system libraries, which are provided by the platform. The executable can then be deployed onto a physical device.

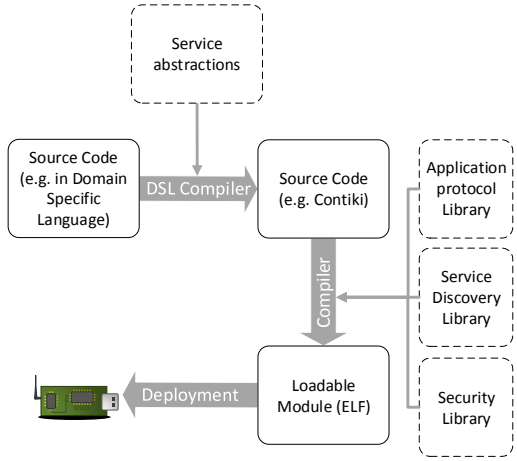


Figure 2.11: Development view of the architecture.

2.6.2.4 Deployment and operational view

Figure 2.12 shows the deployment and operational view of the architecture. It shows how entities are mapped onto devices, and how components in the system interact. Firstly, an application is developed by a user of the system. The user can be a human entity, or be another device (e.g. external system). From the application executable, the system knows which services the application uses, and which software components contain the application logic. Then, the application is installed by placing the software components onto physical devices capable of running them, and activating the components. This is done via the management component of the platform. Finally, upon activation, the application components start accessing the wanted services. This is done either locally, on the same device, or via the network, remotely. In the end, each node may host multiple software components, which belong to different applications, and may use any services available in the network.

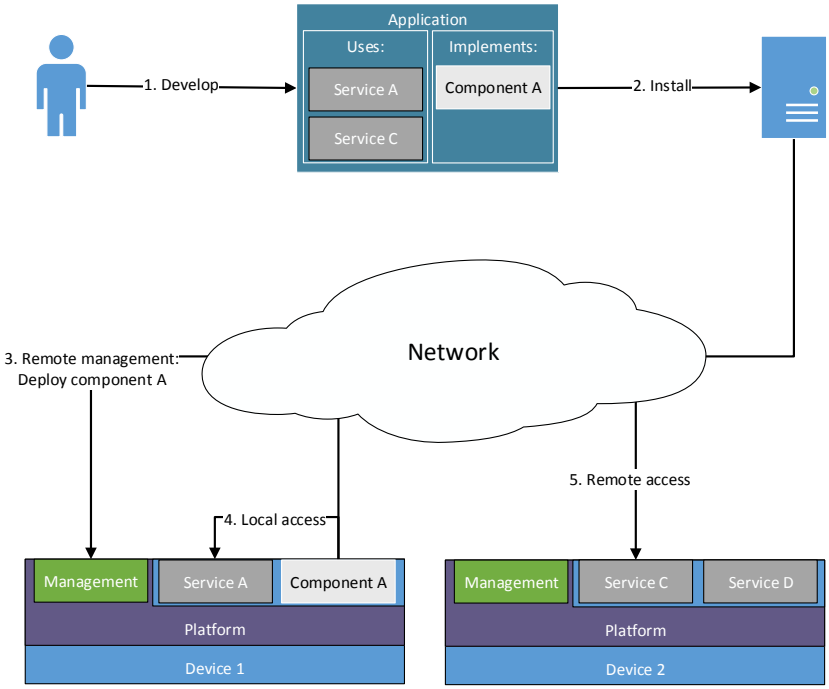


Figure 2.12: Deployment view of the architecture.

2.6.3 Example implementation over IEEE 802.15.4

To demonstrate the applicability of the proposed architecture, we use it to implement one of the previously described IoT use cases - building automation systems, or so called smart buildings. The main concept behind building automation systems is that control over heating, ventilation, lighting and other systems can be done automatically. Such automation is expected to reduce both operational cost and energy usage. From a system perspective [Bot11], building automation systems are challenging due to the large number of devices involved, low cost and low power technology requirements, longevity of the deployment, and the interoperability between various vendors. From end-user perspective [Bru+11], some of the challenges come from the unreliable behaviour of existing systems, limitations in flexibility of already deployed systems and poor management.

The basic physical building blocks of a building automation systems are

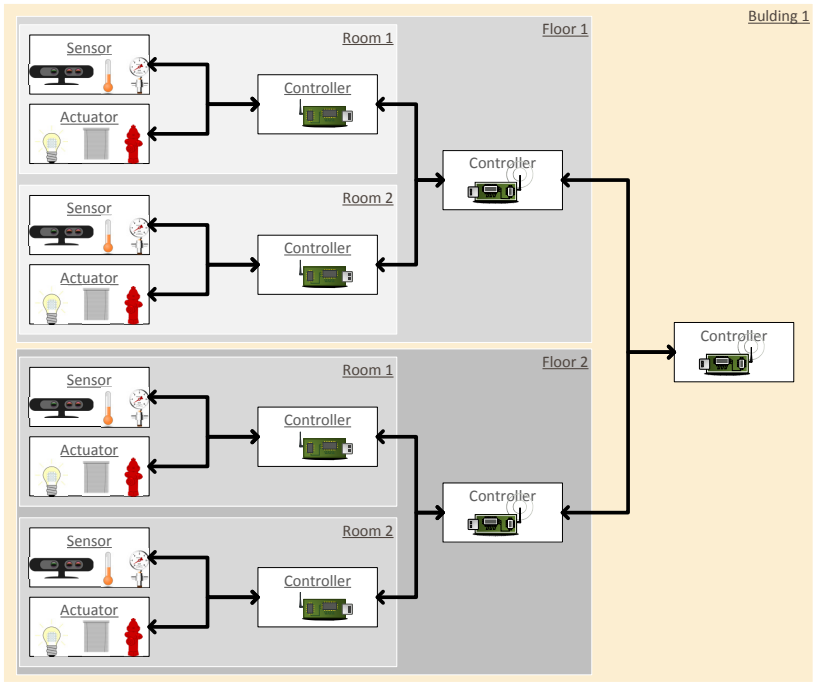


Figure 2.13: Physical view of a smart building.

the various sensors and actuators used to read and control the physical environment (Requirement **R5**). We assume that these are the least powerful devices in the system (constrained class C0 - C2), and are connected wirelessly, using IEEE 802.15.4. Furthermore, these devices have some pre-knowledge of their environment, as their physical and logical location, type of available sensors/actuators, type of power supply, and processing characteristics, stored as *context*.

Beside these fundamental devices, we assume that fewer, more powerful devices (class C3 and above) are present in the network as well. We consider these as controllers, as they have mostly management roles. Controllers can be logically organized in a hierarchical fashion, as in Figure 2.13, even though several logical controllers may be hosted on the same physical device. We use controllers to perform management operation in the network, to control actions between sensors and actuators, and to interface with end users.

Figure 2.14 shows a layered view of the proposed system architecture on top of a standardized protocol suite for IEEE 802.15.4 based LLNs.

Similar architectures have been proposed in [KDD11; Pal+13]. In these architectures, the low-power optimizations are done at the link layer, as part of a separate RDC sub-layer. This separation preserves the layering hierarchy, and allows the application protocol to be built in isolation of the low power behaviour.

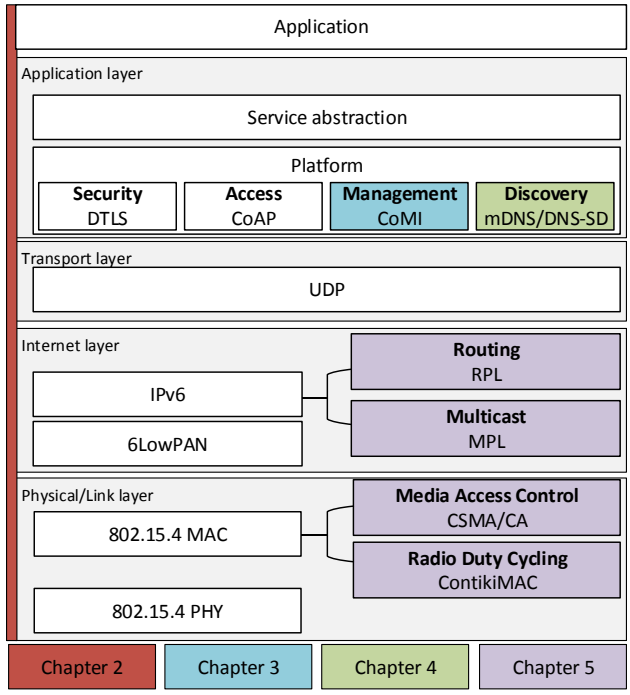


Figure 2.14: Software stack for low-power networks, according to the 4-layer Internet model.

In our specific architecture, we adopt the RESTful approach, with CoAP as the application protocol. RESTful architectures have been shown to be scalable to large number of components and interactions (Requirement **R1**), improve interoperability by defining simple interfaces (Requirement **R4**) and are easily extensible (Requirement **R3**). As previously mentioned, CoAP is lightweight, standardized and ported to many platforms [KDD11] (Requirements **R2**, **R5**). Furthermore, the CoAP specification recommends using Datagram Transport Layer Security (DTLS) as the security mechanism (Requirement **R6**). IETF is also standardizing an integrated management interface - CoAP Management Interface (CoMI) [Sto+15], which we use (Requirement **R3**).

For service discovery, we use the Multicast Domain Name System (mDNS) with DNS-Based Service Discovery (DNS-SD), with several extensions for context-based service discovery and proxy support. mDNS/DNS-SD is a well-known protocol for service discovery in Local Area Networks (LANs), and supports completely distributed operation, without the need of dedicated servers (Requirements **R3**, **R4**). We elaborate the protocol and its extensions in Chapter 4.

We resolve to use an IP based Internet layer, consisting of the IPv6 protocol and the 6LoWPAN adaptation layer. These protocols have been designed to scale to large LLNs (Requirements **R1** and **R2**). Finally, the link layer is specific to low-power operation over IEEE 802.15.4, and consists of unslotted Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) as the Medium Access Control (MAC) protocol, and ContikiMAC [Dun11b] as an RDC protocol. Unslotted CSMA/CA is a probabilistic MAC protocol, which does not need third-party orchestration as the beacon-enabled version. ContikiMAC is an efficient sender-initiated RDC protocol, capable of keeping the radio off on devices for 99% of the time. Both CSMA/CA and ContikiMAC are part of the IEEE 802.15.4e standard, and are able to operate without synchronization between nodes.

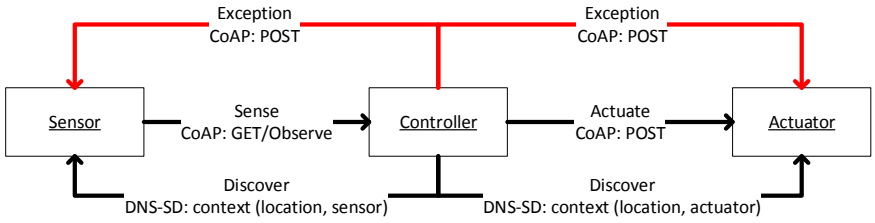


Figure 2.15: Information flow between sensors, actuators and controllers within a smart building.

In this architecture, devices can be programmed both at commissioning time and at runtime. Fundamental devices are usually programmed with the basic services from the platform and any necessary drivers at commissioning time. Controllers are programmed at commissioning time with the platform, but can be re-programmed at runtime at the request of end users. The execution of software components creates a binding between controllers, sensors and actuators, based on type of control and the context properties of the devices. First, the control component discovers relevant services for its functionality based on the

context properties it has. Typically, this context includes the type of service wanted and its physical location. Then, it reads data from a sensor service, processes the input, and controls actuator services. Lastly, it can exhibit basic kind of emergency control over both sensors and actuators, in case of unforeseen circumstances (Figure 2.15).

2.6.4 Open problems

Even though the presented architecture is based on existing protocols, its implementation opens several problems. We address the following ones in the remainder of the thesis.

(Re)-programming a large network of constrained devices. Firstly, the entire system has to be deployed and maintained, as software changes over time. An important aspect is then the initial delivery of system software in the network, i.e. the platform, and all subsequent changes in software components. Due to the constrained nature of the nodes involved, and the massive scale, this is a non-trivial task. We explore this problem in Chapter 3.

Service discovery over large low-power networks. Secondly, it is currently unclear which is the most appropriate protocol for service discovery in the IoT. Solutions for service discovery, as the selected mDNS/DNS-SD have to scale well, have a low memory footprint, and have to manage devices with large access latencies. We conduct a survey of existing protocols, and propose extensions for mDNS/DNS-SD in Chapter 4.

Basic network functionality in large, lossy networks. Thirdly, current networking protocols standardized by the IETF use the Trickle algorithm [LPCS04] for fast and efficient dissemination. Trickle is used in the RPL routing protocol for disseminating routing information, and in the MPL multicast forwarding protocol for disseminating multicast traffic and management information. However, the performance of both protocols is not thoroughly analysed when it is used with low-power radios in different networking topologies. We address these issues in Chapter 5, and propose changes to the Trickle algorithm for improved performance.

2.7 Conclusion

In this chapter, we presented a general system architecture for the IoT, which addresses research questions **RQ1** and **RQ2**. In order to build the architecture, we first analysed IoT use cases and various constraints to proposed solutions. We used the given constraints to evaluate different application design styles for the IoT, as well as existing software protocols operating at various software layers.

We decided to adopt a service oriented system architecture, which is flexible enough to implement any of the different application styles. We described a potential implementation of the system architecture for low-power wireless networks using standards based protocols. Within the given implementation, we identify open research areas which are investigated in depth in the following chapters.

3

SOFTWARE UPDATE

In this chapter, we address a particular problem in the management of large networks of heterogeneous devices (research question RQ3): software update. Software update is an essential feature of any long term deployment in the Internet of Things (IoT), and a core functionality of the system architecture in Chapter 2. Due to the limited network capacity, energy restrictions and low computational capabilities, updating software in Low-Power and Lossy Networks (LLNs) is not trivial.

This chapter improves the state of the art in software updates for LLNs by reducing the size of the updates - smaller updates take less time and less energy to be disseminated. We compare three black-box approaches: directly compressing updates, exploiting version similarities for incremental updates, and exploiting similarities in updates and broadcast-based update schemes to build meta-updates. We verify our approach on two data sets, for software updates of sensor nodes and smart phones.

The chapter is based on the following publications:

- Milosh Stolikj, Pieter J.L. Cuijpers, and Johan J. Lukkien. "Energy-aware Reprogramming of Sensor Networks Using Incremental Update and Compression". In: *Procedia Computer Science 10 (2012)*, pp. 179–187.
- Milosh Stolikj, Pieter J.L. Cuijpers, and Johan J. Lukkien. "Efficient reprogramming of wireless sensor networks using incremental updates". In: *IEEE Conference on Pervasive Computing and Communications Workshops*. PERCOM Workshops. 2013, pp. 584–589.
- Milosh Stolikj, Pieter J.L. Cuijpers, and Johan J. Lukkien. "Patching a patch - software updates using horizontal patching". In: *IEEE Transactions on Consumer Electronics 59.2*. 2013, pp. 435–441.

3.1 Introduction

Software changes over time for various reasons: bugs are being fixed, new features are added, or some pieces of code get completely replaced by others. Therefore, any long term deployment, as are many applications of the Internet of Things, needs to be *reprogrammable*, i.e. have the capability to change software functionality of devices within a network at run time. Due to the massive scale of the IoT, and the lack of direct access to most devices, the only reasonable way for reprogramming is to do it remotely.

Reprogramming is important both during development, for fast prototyping and debugging, and after deployment, for adapting functionality. The frequency of reprogramming depends on the particular use case. For instance, sensor networks need to be calibrated on a regular basis for proper operation. Similarly, dynamic networks, such as home entertainment systems, are frequently upgraded with new features. In both cases, it is important that the reprogramming is swift, since during that period, the network is usually not available for normal operation.

We can categorize reprogramming according to the type of change that is required in the network and on the devices themselves. In general, we call these modifications *updates*. We distinguish the following types of software updates:

- an update of the operating system;
- an update of an application;
- an addition of a new application;
- a modification of parameters in an existing application.

Resource constrained devices in Wireless Sensor Networks (WSNs) are usually programmed in two ways: either by flashing the devices with a complete firmware image, or by loading a partial executable binary. The second approach is more flexible and allows easier extension of applications, without the need to reboot the operating system. Despite its flexibility, it has limited support on existing sensor network platforms.

In both cases, the software update is prepared at an external host, and then distributed in the network, reaching every intended device (Figure 3.1). Hence, while the devices that need to be updated can be

resource constrained, there is no such requirement for the host where the update is prepared. Furthermore, previous research has shown that when many devices in a single network need to be updated, a broadcast-based dissemination scheme is more efficient than a unicast one [SHE03; LPCS04]. In summary, remote software update in LLNs, suffers from four major issues:

- Low-bandwidth, high-latency, and lossy links make the delivery of large data packets difficult.
- Many networks are heterogeneous, and require separate software updates for different node types. In broadcast-based update schemes, as a consequence, all updates would be spread to the entire network (Figure 3.2).
- Nodes can have limited power supply. Since the radio chip consumes considerable energy, often more than the processor, radio transmissions should be kept as low as possible.
- Low processing and storage capabilities render many approaches developed for non-resource constrained devices to be inapplicable.

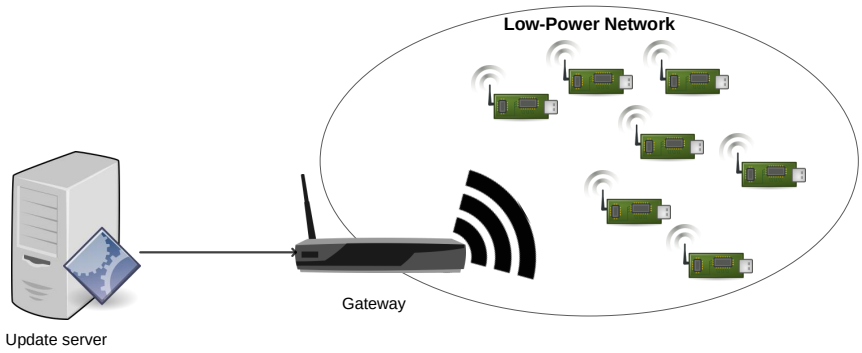


Figure 3.1: Software update in a low-power network. The update is prepared at an outside host, and distributed in the network via a gateway.

In this chapter, we explore means for optimizing broadcast-based remote software updates in LLNs. Based on the previously mentioned issues, as key performance metrics we take:

- Delay for completing a software update.
- Energy consumed for distributing and applying a software update.

- Memory footprint of the involved optimization algorithms.

Our hypothesis is that due to the nature of the distribution network, considerable gains can be achieved at the start of the software update process - by reducing the size of the software updates. With smaller software updates, energy is saved directly, by sending and receiving less data, and indirectly, by keeping the media free, thus reducing the chances for collisions to occur. Many studies have already identified that the radio transmitter as the largest energy consumer within a node [SM06; ZSLM04]. Therefore, we assume that the additional processing is favourable to wireless transmission. We focus on three approaches for reducing the size of software updates:

- Applying data compression algorithms directly on software updates.

- Using incremental updates, i.e. distributing only the difference between two consecutive software versions, captured by delta encoding algorithms in scripts called *deltas*. We refer to this method as *vertical patching*, since the delta is between two different software versions.

- Packing updates of similar devices together, to exploit any similarities in the software of heterogeneous devices and the broadcast-based distribution mechanism. This approach improves the scalability of software updates in large networks. We refer to this method as *horizontal patching*.

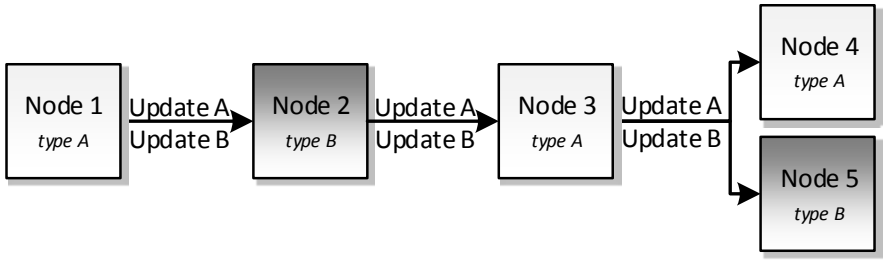


Figure 3.2: Consider a network of multiple devices, which are of one of two types, i.e. run two different software executables. When updating multiple devices of the same type in the same network, broadcast/multicast dissemination is preferred to unicast. However, even though both device types might share components, currently individual updates are prepared for each of them. As a result, the updates for both device types will be broadcasted to all devices.

We start with five data compression algorithms and three algorithms for delta encoding. We take these algorithms as black boxes, without modifying their inner workings. We first port these algorithms to resource constrained devices, and profile their size and memory requirements. Then, experimentally, we test how much we can reduce the size of the updates by applying each data compression algorithm alone, or in combination with the three delta encoding algorithms. As a test set, we use several types of updates: an update of the operating system, parameter change in an application and a major update of an application. Our results show that using only data compression can degrade performance, while delta encoding always gives improvements. The improvements depend on the type of update and the algorithms involved, but we observed a minimum of 50% reduction in size with certain combinations.

Finally, we present a new algorithm for reducing the size of updates when multiple devices sharing a common platform need to be updated, called *horizontal patching*. Instead of distributing separate updates for each device type (i.e. distribute all *vertical deltas* per device type), we distribute one update for one application type (i.e. one vertical delta as a basis), and create *horizontal deltas* for the rest, for rebuilding an update for a different device type from the initial update. One hopes that when the common platform is updated, all updates hold the same information, and there is a large similarity between them. Therefore, the horizontal deltas should be smaller in size than the sum of all individual updates.

We verify our approach on two test cases. The first test case consists of firmware updates for sensor nodes in a LLN, with 7 different applications for the Contiki operating system [DGV04], when the operating system is updated. The second test set consists of firmware updates of Android-based devices. Our results show between 10% and 30% improvement of horizontal patching over using only vertical deltas, which is significant.

This work addresses requirements **R1**, **R2**, **R3** and **R5**, defined in Section 2.3.5. Firstly, the horizontal patching approach improves scalability of broadcast-based software updates in multi-application networks (Requirement **R1**). Secondly, the reduced energy usage contributes to the applicability of software updates in constrained networks (Requirement **R2**). Thirdly, the reduced dissemination time improves the responsiveness of the system, and reduces the maintenance time (Requirement **R3**). Finally, the small implementations of the algorithms contribute to the portability of the approach (Requirement **R5**).

This chapter is structured as follows. In Section 3.2 we give an overview on related work on software update in resource constrained environments. Then, in Section 3.3 we describe how data compression and delta encoding can be included when preparing software updates, and we describe the horizontal patching approach. We evaluate all approaches in Section 3.4, and give concluding remarks in Section 3.5.

3.2 Related work

Optimizing software updates has been well studied within the research community. Many approaches have been developed over time, from general purpose methods, such as incremental updates, to device specific optimization routines. In this section, we first cover generic algorithms for incremental update. Then, we consider their implementation for updating both consumer electronic devices and resource constrained devices as part of WSNs. Lastly, we present studies on updating multiple software versions.

3.2.1 Incremental update

Methods for incremental update use delta encoding, or DIFF algorithms, for extracting the difference between two consecutive software versions. A rough classification of these algorithms can be made based on the type of matching done between software versions. One group of algorithms, including VCDIFF [RFC3284] and RSYNC [Tri00], finds completely identical blocks between two consecutive software versions. This makes them relatively fast during both delta generation and patching. The second group of algorithms, such as BSDIFF [Per03], find similar but not completely identical data blocks between two software versions, and encode the differences. Deltas generated using this approach are generally smaller, but take more time to be created.

The main drawback of general-purpose DIFF algorithms is that they are resource intensive. As a result, they have been largely avoided in resource constrained environments. However, due to the asymmetric nature of software updates, as we show in section 3.3.2, this is not always a rightful decision.

3.2.2 Incremental update in consumer electronic devices

Algorithms for incremental update are general enough to have been applied in many domain-specific applications. This ranges from software updates in mobile phones [KM10], on-vehicle information devices [KTT12], and sensor networks. Domain-specific variations of delta encoding algorithms [PBM11; SC12; JC04] have been built to enhance the delta generation process in order to further reduce the update size. These algorithm adaptations can be seen as best practices, which can be transferred to other domains to reap similar benefits.

3.2.3 Software update in WSNs

Early software for WSNs was built in a non-modular fashion, where both the application and the operating system were packed in one firmware image. In such systems, updates of any of the components requires a complete change of the firmware image. Modular systems are an improvement over non-modular systems by supporting dynamic linking and loading. This way, operating systems such as Contiki [DGV04], allow partial executables (ELF files) to be deployed and executed at run time, without flashing the firmware. However, since the partial executables contain symbol and relocation tables, they can still be large in size for transfer in lossy wireless networks.

Apart from complete firmware reprogramming [CT03; LPCS04], alternative methods have been developed for updating non-modular systems. Virtual machines and middle-ware layers (Maté [LC02], Open Service Architecture for Sensors (OSAS) [BLV09]) overcome limitations of large updates for distribution by running interpreted code. Since byte code is much smaller compared to compiled binary code, updates in these systems can be easily distributed. The downside of this approach is that interpreted execution is slower and some resources are always used by the virtual machine. Moreover, the problem of large updates is still present if the operating system or the virtual machine engine need to be updated.

Another approach to reprogramming is to use incremental updates of firmware images [MP10]. In [JC04], modified versions of the rsync and XNP protocols are used for generating deltas and their dissemination, respectively. Zephyr [PBM11] adds application-level modifications to

decrease the difference between consecutive application versions, then produces deltas with *rsync*. In [RL03], a tool similar to the *UNIX diff* is used to create deltas between versions. It extends the delta functionality with two new instructions, which enable more efficient coding of the differences. While these studies emphasise the benefits of using incremental updates, they use solely one algorithm, without evaluating whether a better option exists. As we show further on, combining incremental update algorithms with data compression algorithms gives a broader scope of options for optimizing the size of updates.

Data compression has been previously considered in sensor networks, mostly for data gathered from sensors [MV09; SM06]. In [DB10], several algorithms are compared on desktop machines, for compressing data from two test beds. Similarly, in [TDV08] compression algorithms are compared on ELF executables for the Contiki operating system. Since the compressibility of sensed data differs from binary data, the reported results do not apply to software updates, which we address in this chapter. Furthermore, during upgrades, only decompression is needed on resource constrained devices.

3.2.4 Multi-version software update

Related work on updating multiple software versions mainly focuses on incremental updates of a single application. In [KTT12], the authors describe a method for merging multiple consecutive VCDIFF deltas for one application to decrease the cumulative delta size. The result is a single delta which contains instructions and data to build the latest software version from any of the previous ones. The work presented in this chapter is complementary, focusing on situations where multiple applications need to be updated.

In [BRFB11], an epidemic propagation protocol to handle the distribution of multiple deltas of applications for mobile operating systems is described. The protocol assumes that a single application can evolve into multiple orthogonal versions, hence multiple deltas exist for it. Their approach optimizes the gathering of deltas in an opportunistic fashion. Finally, in [SH11], offline planning of updates of multiple applications is proposed. The planning is done according to the combination of deltas that has the smallest size. The work presented in this chapter broadens the scope of the last two studies, by allowing one delta to be the source of another delta, essentially expanding that search space.

3.3 Optimizing software updates

Consider a network of multiple, heterogeneous nodes sharing a common software component, such as an operating system, software middleware or virtual machine engine. The software stack on each device consists of a set of applications running on top of the shared software component. In certain systems, e.g. a sensor network, the applications are bundled and distributed together with the shared software component. In such systems, an update of the shared software component results in an update of the entire bundle. The bundle of an application with the shared software component defines a software entity for update, and we refer to it as the *software bundle* for a specific device type.

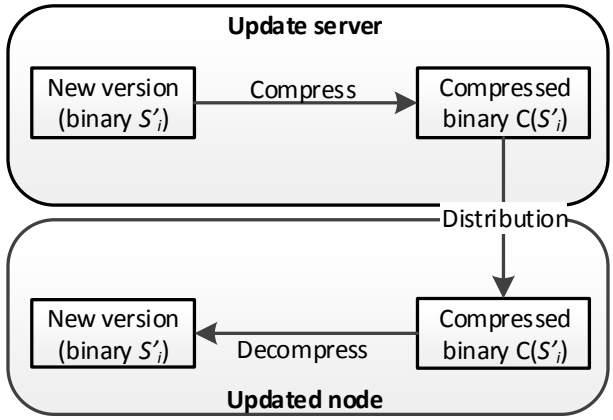


Figure 3.3: Overview of the update process when using data compression.

3.3.1 Data compression algorithms

Let $S = S_0, S_1, \ldots, S_{n-1}$ be the old version of the software bundle for device type $i, i = 0, 1, \ldots, n-1$. The new version of the software bundle is then $S' = S'_0, S'_1, S'_2, \ldots, S'_{n-1}; i = 0, 1, \ldots, n-1$. A simple method to reduce the size of data for transmission for update is to compress the new software bundle before distribution:

$$C(S'_i) = \text{compress}(S'_i), i = 0, 1 \ldots, n-1. \tag{3.1}$$

Compression, and accordingly decompression, is added to the update process as shown in Figure 3.3. It is an intermediate phase with the aim to encode information with fewer bits than the original representation. Data compression is done outside of the network, so only decompression is needed on the updated devices. Furthermore, since in executable data every bit is equally important, only lossless compression algorithms can be used. While many data compression algorithms are available, most of them are inapplicable on resource constrained devices due to high resource demands.

Based on previous research [TDV08; SCL12b; DB10], we selected five Lempel-Ziv [ZL77] based compression algorithms for comparison. These studies concluded that Lempel-Ziv algorithms provide a good trade-off between compression performance and computational complexity.

Lempel-Ziv algorithms maintain a look-up dictionary of frequently seen symbol sequences. Whenever a match is found in the uncompressed data, it is replaced with a reference to the dictionary. Several flavours of the algorithms are interesting. The initial variant of the algorithm, Lempel-Ziv 77 (LZ77), uses a sliding window of previously seen data as a dictionary, and references can point anywhere in it. Fast Lempel-Ziv (FastLZ) [i] and Lempel-Ziv-Jeff-Bonwick (LZJB) [ii] are based on the LZ77 algorithm, with improvements for speed. Run-Length Encoding (RLE) can be seen as a special variant of LZ77, where the sliding window has a length of one symbol. Finally, Lempel-Ziv 78 (LZ78) is a variant where an explicit lookup dictionary is constructed and maintained. We use Sensor Lempel-Ziv-Welch (S-LZW), a data compression algorithm specifically designed for sensor data, as a representative of LZ78 algorithms.

3.3.2 Delta encoding algorithms

Additional reductions in the size of the updates can be achieved by exploiting the similarities between the *old* and *new* version of the software bundle. Since most updates are incremental, the consecutive software versions share most of the code base, and the difference between them is significantly smaller than the size of the bundle itself.

Algorithms for delta encoding exploit this behaviour by extracting and distributing only the differences between both versions in scripts called

[i] http://www.fastlz.org/
[ii] http://en.wikipedia.org/wiki/LZJB

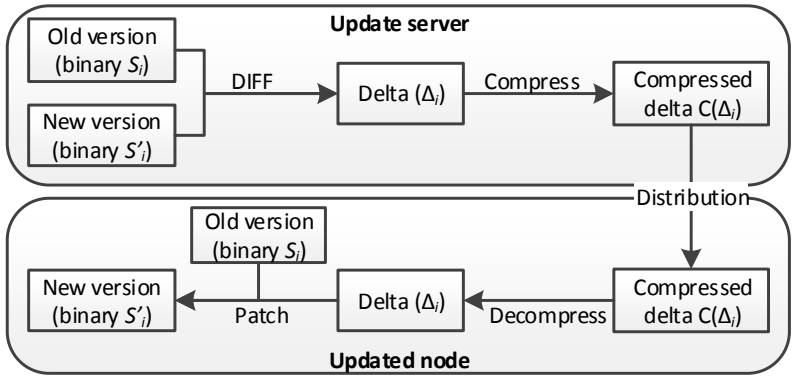


Figure 3.4: Overview of the update process when using incremental updates.

deltas (Δ). Deltas contain instructions and data for reconstructing the new version from the old one, through a method called *patching*. To further reduce size, the delta is compressed ($compress(\Delta)$) before distribution.

Delta encoding algorithms differ in how the delta is constructed and how the differences are detected. Similar to data compression, the delta creation is done outside of the sensor network. Therefore, on the updated devices, only algorithms for decompression and patching need to be implemented (Figure 3.4). Since these deltas are used to transform different versions of the same software set, we refer to them as *vertical deltas*, defined as:

$$\Delta_i = \text{diff}(S_i, S'_i), i = 0, 1, \ldots, n-1. \tag{3.2}$$

Next, we give a brief overview of the three most popular delta encoding algorithms. For all examples, we use the strings in Figure 3.5 as the old/new data.

Old data: ABBBBACBBBAABBABBBBCBBBAABABBA

New data: ABBBBCCBBBAABABBBBCBBBAABACCFA

Figure 3.5: Sample input data for the delta encoding algorithms. The red letters are the modified information in the new data, while the green letter is the new information.

3.3.2.1 BSDIFF Delta Encoding

BSDIFF [Per03] is a well-established algorithm for delta encoding. BSDIFF has a two-pass algorithm to construct optimized deltas. In the first pass, completely identical blocks are found in the two versions. Next, these blocks are extended in both directions, such that every prefix/suffix of the extension matches in at least half of its bytes. These extended blocks correspond to the modified code.

The BSDIFF delta is built of three parts (Figure 3.6): a control block of commands; a diff block of bytewise differences between approximate matches and an extra block of new data. When the old and new versions are similar, the diff block consists of large series of zeroes, which are easy to compress.

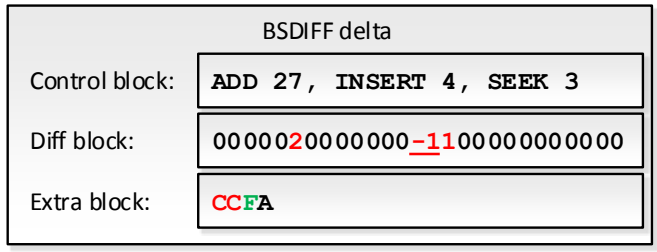


Figure 3.6: Example of a BSDIFF delta. ADD specifies that the first 27 bytes from the old data and from the Diff block are summed. Zeroes in the Diff block mean that the corresponding byte from the old data is unchanged. INSERT adds four bytes from the Extra block to the output. SEEK moves the pointer in the old data three places forward, to the end of the stream.

3.3.2.2 VCDIFF Delta Encoding

VCDIFF [RFC3284] is a format for encoding the difference between two data sets (Figure 3.7). The original idea for VCDIFF comes from data compression algorithms - the old and new version are concatenated; then the resulting stream is compressed using a data compression algorithm. From the output, the first part, which corresponds to the old version, is omitted, leaving only the instructions for the decoder to decompress the new version.

VCDIFF features a detailed byte-code instruction set, consisting of a small number of instructions, which can be used in different addressing

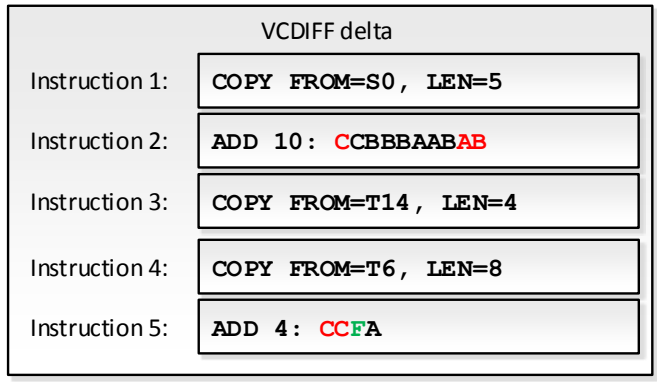


Figure 3.7: Example of a VCDIFF delta. The first instruction copies the first 5 bytes from the old data (S0). Then, the next 10 bytes are added, after which two blocks from the newly written data are copied (T14 and T6). The last four bytes are again added from the delta.

modes, accessing both the old and the new data. Additionally, a cache of recent addresses is held in memory.

Several tools for generating VCDIFF deltas are available. In this work, Xdelta [iii] is used as an encoder for generating VCDIFF deltas. It uses additional heuristics for optimizing the generated instruction set, such as removing completely covered instructions and combining small instructions into one, reducing the delta size.

3.3.2.3 RDIFF Delta Encoding

Rsync, and the corresponding RDIFF algorithm [Tri00], use non-overlapping fixed-sized blocks for matching identical data between the old and new version (Figure 3.8). Both versions are segmented into blocks, and for each block, a rolling-checksum and a MD5 checksum are computed. Based on these checksums, the delta is constructed of either references to blocks that already exist in the old version, or the entire content of new or changed blocks.

A weakness of the algorithm is that if two blocks differ in even one byte, the entire block content has to be present in the delta. Finally, while the rolling checksum is implemented to be as fast as possible, an MD5 checksum is not appropriate for resource constrained devices.

[iii] http://xdelta.org/

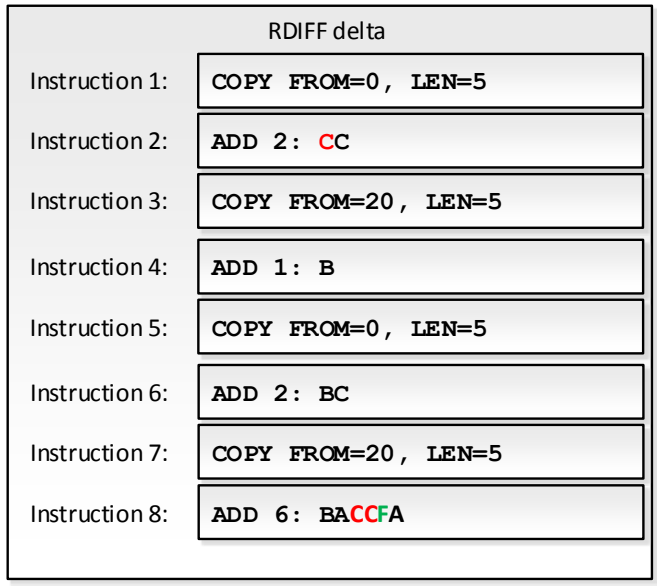


Figure 3.8: Example of a RDIFF delta, using blocks of 5 bytes. The RDIFF delta copies identical blocks of 5 bytes from the old data, and inserts everything in between with ADD instructions.

3.3.3 Horizontal Patching

Vertical deltas are not universal: a delta created for one application on a certain platform cannot be applied on a different application or a different platform. Therefore, updating multiple devices in a network would require distributing each of the individual deltas, as shown in Figure 3.2.

Horizontal patching is a way to reduce the size of data that needs to be distributed in the network. When a shared component is updated, all vertical deltas essentially hold the same information. Therefore, it is possible to use one vertical delta as a basis, and generate other deltas from it (Figure 3.9):

$$\delta_{i,j} = \text{diff}(\Delta_i, \Delta_j); i, j = 0, 1, \ldots, n-1. \tag{3.3}$$

Since both deltas hold the same modifications, the horizontal delta between them should be smaller in size than any of the vertical deltas. The combined delta then consists of the basis and the horizontal deltas. For

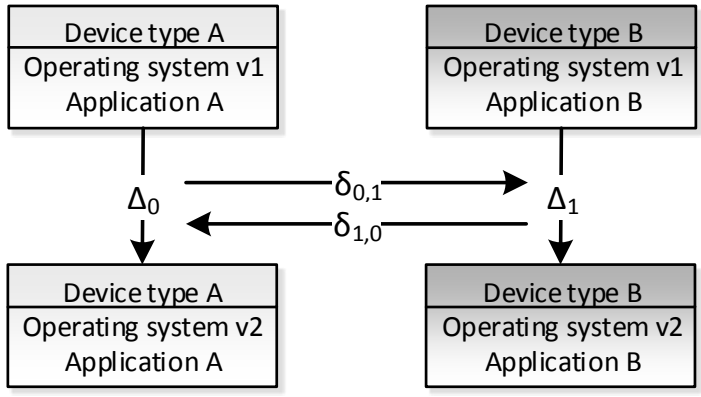


Figure 3.9: Possibilities for horizontal patching in a two-application network. The two devices share the same operating system.

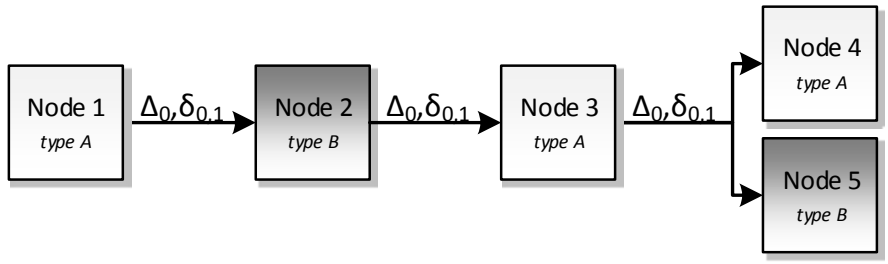


Figure 3.10: Horizontal patching in practice. Both the basis vertical delta (Δ_0) and the horizontal delta ($\delta_{0,1}$) are broadcasted to all devices in the network. On devices of type A, only Δ_0 is used for patching. On devices of type B, first Δ_1 is built by applying patch $\delta_{0,1}$ on Δ_0. Then, Δ_1 is used to patch the system.

the example in Figure 3.9, two vertical (Δ_0 and Δ_1) and two horizontal deltas are possible ($\delta_{0,1}$ and $\delta_{1,0}$). Then, the combined delta can consist of Δ_0 + $\delta_{0,1}$ or Δ_1 + $\delta_{1,0}$. E.g., when Δ_0 and $\delta_{0,1}$ are used, only Δ_0 needs to be executed for updating devices of type A. On devices of type B, first $\delta_{0,1}$ is executed on Δ_0, producing Δ_1; finally, Δ_1 is executed (Figure 3.10).

All algorithms for incremental update use some form of compression to reduce the size of the vertical deltas. Unfortunately, due to the relocation and in some cases, obfuscation, introduced by this compression, it is very difficult to compute efficient horizontal delta directly on compressed vertical deltas. Therefore, we compute the horizontal deltas on uncompressed vertical deltas, and afterwards compress the combined vertical delta with horizontal delta(s) for distribution.

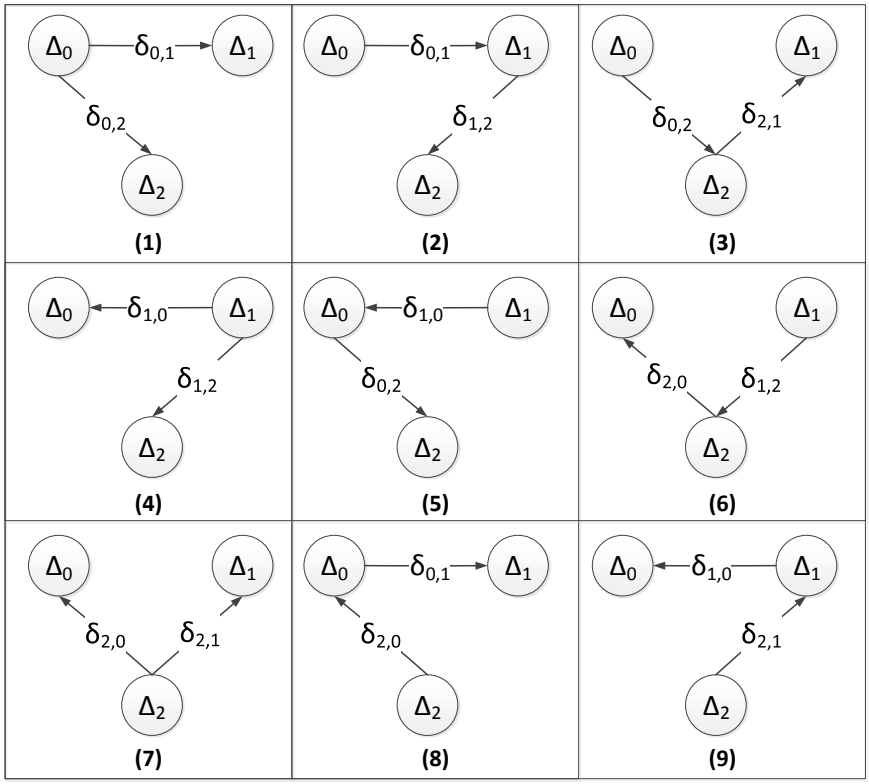


Figure 3.11: Nine different options (minimal spanning trees) for horizontal patching of three heterogeneous devices.

3.3.3.1 Scalability

The number of horizontal deltas rapidly grows as the number of device types increases. Then, selecting the best option for horizontal delta can be seen as finding a minimal spanning tree in a labelled di-graph (Figure 3.11). Each vertex in the di-graph represents a vertical delta, whereas each edge corresponds to a horizontal delta. The cost of each edge is equal to the size of the associated horizontal delta. According to Cayley's formula [Cay89], the number of spanning trees on n labelled vertices is n^{n-2}. For the number of possibilities for horizontal patching, this value needs to be multiplied by n, for each vertical delta as the base. Therefore, choosing an optimal horizontal delta would result in searching for the minimal cost spanning tree from n^{n-1} possible trees.

Algorithm 1 Greedy search of a horizontal delta.

Input: N - number of vertical deltas, Δ - array of sizes of vertical deltas, δ - matrix of sizes of horizontal deltas.
Output: greedy horizontal delta combination of the given di-graph.

```
1: reachable ← {};
2: path ← {};
3: s ← argmax (Δ);
4: append(reachable, s));
5: append(path, new edge (s, s, Δ_s));
6: while len(reachable) < N do
7:     min ←MAX_INT;
8:     for all i ∈ reachable do
9:         for j = 0 to N − 1 do
10:            if j ∉ reachable and δ_{i,j} < min then
11:                min ← δ_{i,j};
12:                from ← i;
13:                to ← j;
14:            end if
15:        end for
16:    end for
17:    append(reachable, to);
18:    append(path, new edge (from, to, min));
19: end while
```

The processing time for finding a minimal spanning tree can quickly explode as the number of types of devices increases. This is due to the large number of possible paths which have to be searched, as well as the processing time required to create the entire di-graph. The time required for building a delta depends on the size of the old and new versions. For example, BSDIFF creates a delta in $O((x+y)logx)$ time, where x is the size of the old version and y is the size of the new version. Therefore, the processing time can be extremely long for large input, such as firmware images for smart phones, ranging from a few hours to several weeks with existing computing resources. Building a complete di-graph would require computing $n(n-1)$ horizontal deltas in addition to the n vertical deltas, which can become excessively long.

In order to reduce processing time, the number of considered deltas has to be reduced. An easy way to approach this problem is to greedily

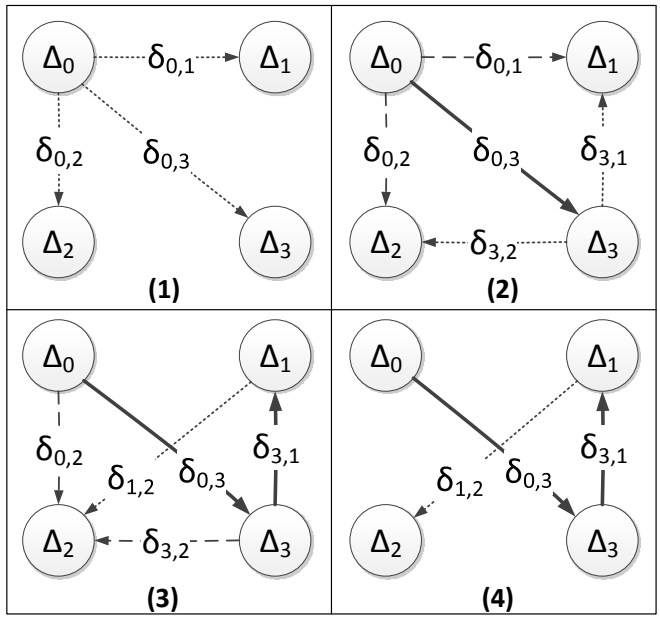


Figure 3.12: Greedy search of a horizontal delta. Assuming that Δ_0 is the largest vertical delta, in the first iteration the three horizontal deltas are inspected. In (2), after $\delta_{0,3}$ is chosen, the di-graph is expanded with edges from Δ_3 which reach new vertices ($\delta_{3,1}$ and $\delta_{3,2}$). The edges towards unreachable vertices from the previous step are also taken into consideration ($\delta_{0,1}$ and $\delta_{0,2}$). After $\delta_{3,1}$ is selected (3), one more edge is computed ($\delta_{1,2}$). By adding $\delta_{1,2}$, all vertices are reachable. The horizontal delta then consists of Δ_0, $\delta_{0,3}$, $\delta_{3,1}$, $\delta_{1,2}$.

build the tree, using the minimum number of edges for comparison. The greedy algorithm, shown in Algorithm 1, expands the tree from the largest vertical delta. The largest vertical delta is chosen as a root because in horizontal deltas, it takes less bits to omit data than to add new data. As a result, horizontal deltas from larger to smaller vertical deltas are most likely to be smaller in size than the corresponding deltas in the opposite direction. In each iteration, a new edge is selected based on two criteria: a) it connects a new vertex; b) no other edge exists such that it connects a new vertex and it is smaller in size than the selected edge. An example of the execution of the algorithm is shown in Figure 3.12. The greedy approach requires $\frac{n(n+1)}{2}$ deltas for computation, which hopefully is feasible to compute. The performance of the greedy algorithm compared to the minimal cost spanning tree is evaluated in the next section.

3.4 Evaluation

In order to verify the impact of data compression, delta encoding and horizontal patching, we perform several experiments. We first define the relevant metrics for comparison. Then, we describe the experiments for comparing the different algorithms for data compression and delta encoding. Lastly, we demonstrate how horizontal patching compares to using only vertical patching.

3.4.1 Metrics

We select four metrics for comparison: code size of the algorithm, memory used during execution, energy and delay. The size of compressed data and execution time are two additional factors which directly determine energy usage and delay. In our experiments, we measure the size of the data produced by the different approaches offline, while the updates are created. We profile the code size, memory consumption and execution time directly on hardware. For the energy usage and delay, we resort to estimation using models presented in this section.

The effectiveness of compression algorithms is usually quantified through the compression ratio. It is defined as the reduction in size relative to the uncompressed data:

$$compr\_ratio = (1 - \frac{compressed\_size}{uncompressed\_size}) \cdot 100 \tag{3.4}$$

Consequently, higher values mean smaller compressed files, hence better performance.

Decompressing data requires a certain amount of processor cycles. A high number of processor cycles would result in larger energy consumption and larger execution time. Therefore, this value should be as low as possible. The importance of this metric is captured through the energy and delay models.

Memory is limited in resource constrained devices. This includes both memory required for holding the code, which is stored in internal flash memory (ROM), and memory required during execution, in RAM. Algorithms running on sensor nodes must have a small code footprint, up to a couple of kilobytes, and use little memory during execution.

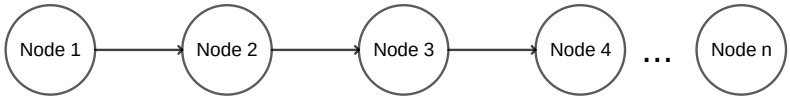


Figure 3.13: Topology for estimating the update delay and energy consumption.

We estimate energy usage through a model which uses the amount of time spent during computation and transmission of data [ALV10]. This is a lower bound of the real energy usage; we assume that forwarding is done immediately, without additional processing, nodes are synchronized, and transmission and reception occurs simultaneously, and we ignore MAC protocol behaviour. Adding those variables, will result in higher energy usage for transmission, penalising communication even further.

We assume that the topology is fixed, and all nodes are arranged in a line (Figure 3.13). The update is spread from the left most node, and every subsequent node first receives the update, then forwards it, and finally applies it locally. The right-most node does not do any additional forwarding. We calculate overall energy usage as:

$$E = k_{\texttt{err}} \cdot \lceil \frac{data\_size}{payload\_size} \rceil \cdot (h-1) \cdot (E_{\texttt{rx}} + E_{\texttt{tx}}) + h \cdot E_{\texttt{cpu}}, \qquad (3.5)$$

where h is the number of nodes in the network, $k_{\texttt{err}}$ is the average number of times each packet is sent due to errors in the radio medium, $data\_size$ is the size of the data for transmission, $payload\_size$ is the maximum packet size, $E_{\texttt{rx/tx}}$ is the energy required to receive/send one packet and E_{cpu} is the energy required for post-processing. Communication energy is expressed as $E_{\texttt{rx/tx}} = t_{\texttt{rx/tx}} \cdot I_{rx/tx} \cdot V$, where $t_{rx/tx}$ is the amount of time that the wireless radio is in listening/sending state. We simplify the model by assuming that during reception, the radio chip is turned on for the same amount of time as during sending, though it draws more current ($t_{\texttt{tx}} = t_{\texttt{rx}}$) [Jen06]. This corresponds to factory values of various radio chipsets, such as the CC2420. Similarly, processing energy is calculated as $E_{\texttt{cpu}} = I_{\texttt{cpu}} \cdot V \cdot t_{\texttt{cpu}}$, where $t_{\texttt{cpu}}$ is the amount of processing time.

We estimate the time needed to update all nodes in the network, i.e. the update delay, with a similar model to the one used for energy estimation. Again we estimate a lower bound of the delay, since we assume that

forwarding is done immediately, the MAC protocol does not introduce additional overhead:

$$D = k_{\texttt{err}} \cdot \lceil \frac{data\_size}{payload\_size} \rceil \cdot (h-1) \cdot t_{\texttt{rx}} + t_{\texttt{cpu}}. \tag{3.6}$$

We consider three cases of energy usage and delay during reprogramming: 1) neither compression nor incremental updates is used ($t_{\texttt{cpu}} = 0$); 2) only compression is used ($t_{\texttt{cpu}} = t_{\texttt{dcmp}}$) and 3) both compression and incremental update is used ($t_{\texttt{cpu}} = t_{\texttt{dcmp}} + t_{\texttt{patch}}$).

Table 3.1: Test cases and data size of firmware images and ELF executables (in bytes).

Test	Description	Type	Old version	New version
1	Operating system update (Contiki 2.3 → 2.4)	Firmware	22,924	20,624
2	Operating system update (Contiki 2.4 → 2.5)	Firmware	20,624	22,980
3	New application (OSAS 2.0)	Firmware	22,980	39,112
4	Application update (OSAS 1.0 → 2.0)	Firmware	37,796	39,112
5	Application update (OSAS 1.0 → 2.0)	ELF executable	25.784	26.712
6	Parameter change (OSAS 2.0 → 2.1)	Firmware	39.112	39.112
7	Parameter change (OSAS 2.0 → 2.1)	ELF executable	26,712	26,712

3.4.2 Data compression and incremental updates

We consider seven test cases for software update, shown in Table 3.1. The first two test cases represent firmware updates of the operating system. The third test case is the replacement of a firmware image with a larger one, which contains a completely new application. The fourth and fifth test case are firmware/ELF updates of an application. The last two test cases are a minor change inside an application (2 bytes), but which result in a new software version.

For each test case, both the initial version and the new version are available. First, we compress the new version directly. Then, we produce

a delta using each delta encoding algorithm, and compress it. We measure the compression ratio of the compressed delta with respect to the size of the uncompressed new version. Finally, we measure the remaining metrics for decompression and patching.

We use the Contiki operating system, running on Crossbow TelosB nodes (also known as Tmote Sky) [PSC05], with the OSAS [BLV09] application. The node contains an 8 MHz TI MSP430 microcontroller with the Chipcon CC2420 radio transceiver. It has 48 kB program flash memory, 10 kB of RAM and 1 MB external flash. All algorithms are ported for TelosB nodes [iv]. Input and output data is stored on the external serial flash and is accessed through the Coffee file system [TDHV09]. All tests are executed 10 times, and timed using the Contiki clock module.

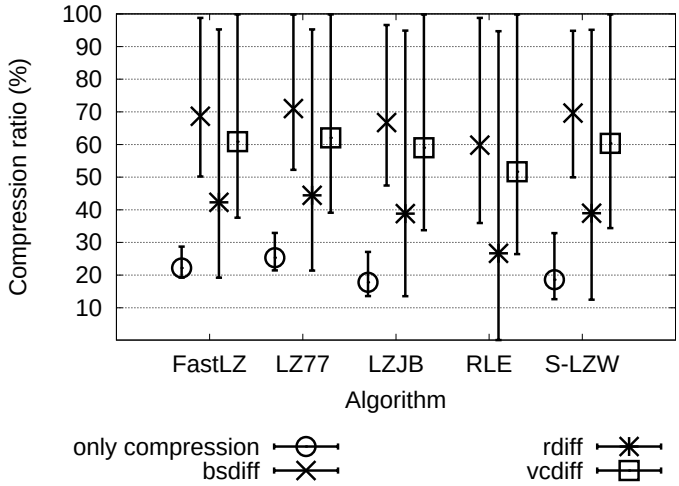
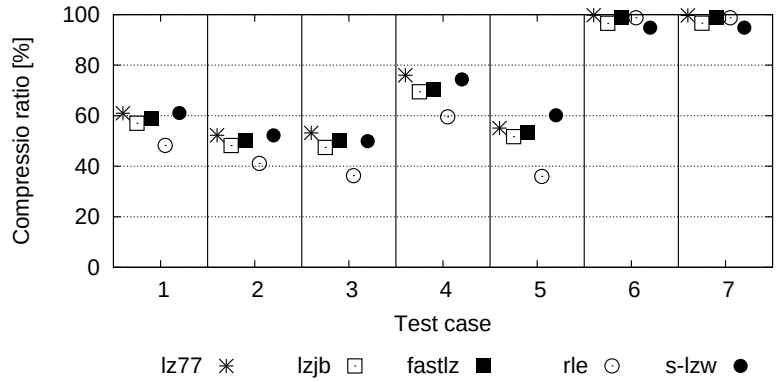


Figure 3.14: Minimum, maximum and average measured compression ratio of test cases 1-7.

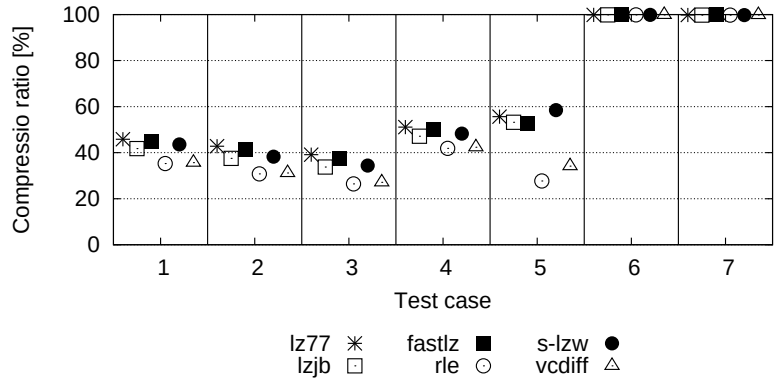
3.4.2.1 Compression ratio

Compression ratio is a factor which gives a strong indication what to expect from a compression algorithm in terms of energy and delay savings. However, it is highly dependent on the type of input data. As illustrated in Figures 3.14 and 3.15, due to the diverse input samples, the compression ratio varies significantly between different test cases.

[iv] The VCDIFF implementation was kindly provided by Nicolas Tsiftes [TDV08]



(a) BSDIFF



(b) VCDIFF

Figure 3.15: Compression ratio of different compression algorithms when used in combination with BSDIFF (a) and VCDIFF (b), per test case.

The first two tests cases, which demonstrate an update of the Contiki operating system, give an estimate on the amount of change in the subsequent software releases. While direct compression on the resulting firmware images shows similar results, incremental updates from version 2.3 to 2.4 are approximately 10% smaller than from version 2.4 to 2.5. This is a clear indication that version 2.5 of the Contiki operating system is a major update to version 2.4, unlike 2.4 to 2.3. Similarly, the compression ratio for test case 3 is lower than the first two test cases because the new firmware image is twice as large as the old one. Most of the data in the new firmware image is not seen before, and has to be inserted.

The compression ratio of test cases 4 and 5 is noticeably higher than in the previous scenarios. This is due to the fact that the difference between the OSAS versions is not as significant as the operating system updates. Thus, the deltas can be compressed better.

In the last two test cases, since the difference in the data is only in two bytes, both delta encoding algorithms are then able to pack the entire update in a single packet, resulting in very high compression ratio.

Figure 3.14 shows aggregate information of the compression ratio of the different compression algorithms when used individually, or with the different delta encoding algorithms. The general observation is that incremental updates make significant difference in the performance of compression algorithms. Depending on the approach and type of updates that need to be compressed, we observe between 37% and 99% compression ratio. Most compression algorithms behave similarly, with not more than 10% difference between them. The obvious exception is RLE as the worst compressor.

Using BSDIFF shows higher compression ratio compared to the other delta encoding algorithms in all except the last two scenarios, in which VCDIFF produces smaller deltas (Figure 3.15). Since RDIFF is consistently inferior to the other two algorithms, we omit it from the subsequent experiments.

3.4.2.2 Memory requirements

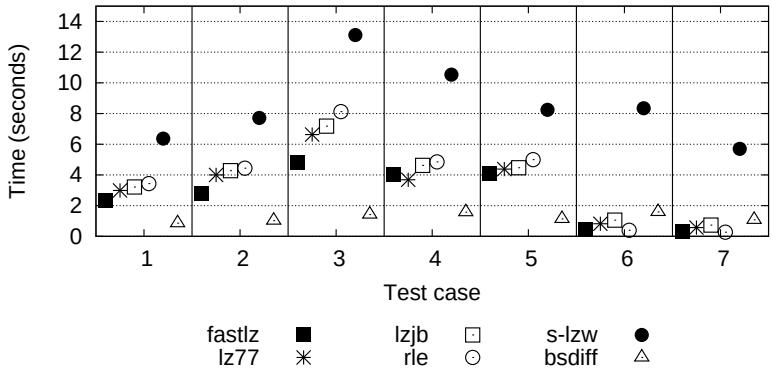
Table 3.2 shows code size and memory usage for the decompression and delta encoding algorithms. The code size corresponds to the size of the .text segment of the ELF binary. Memory is the sum of static memory and maximum stack space used during execution.

From the table, it is evident that RLE, LZ77, and LZJB are lightweight in terms of both code size and memory usage during execution; FastLZ has a larger code base, but still uses little memory. Finally, S-LZW has the largest code base and uses the most memory of all algorithms.

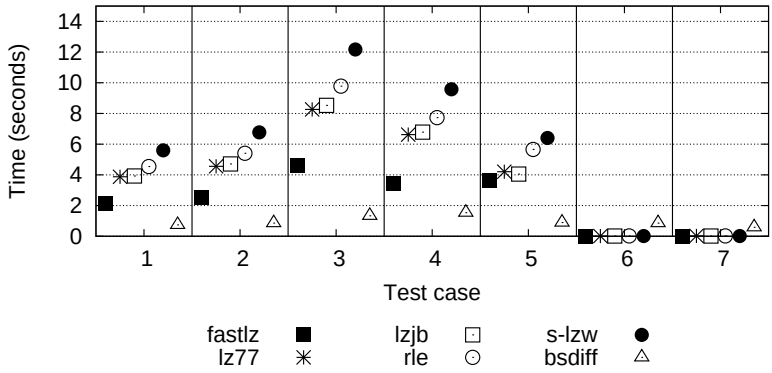
The memory footprint of BSDIFF is small, both in code size and memory usage. On the other hand, VCDIFF has a significantly larger code base, and large memory footprint, mostly for storing the instruction cache.

Table 3.2: Code and memory footprint of different algorithms.

Algorithm	Code (bytes)	Memory (bytes)
fastlz	878	145
lz77	376	144
lzjb	424	140
rle	198	131
s-lzw	1.281	2502
bsdiff	560	158
vcdiff	2.261	1714



(a) BSDIFF



(b) VCDIFF

Figure 3.16: Time required for decompressing and applying a BSDIFF (a) and VCDIFF (b) patch.

3.4.2.3 Processing requirements

The time required to decompress the BSDIFF/VCDIFF deltas on TelosB nodes is shown in Figure 3.16. In all cases, S-LZW is the slowest of all algorithms. LZ77 and LZJB have similar execution times, while RLE has significantly worse performance while decompressing VCDIFF deltas. This comes down to the nature of the VCDIFF algorithm - RLE is already included while the delta is generated. Finally, on average, FastLZ is the fastest algorithm.

BSDIFF and VCDIFF have comparable execution time. VCDIFF is slightly faster in the last two test scenarios (parameter change), due to the smaller delta produced.

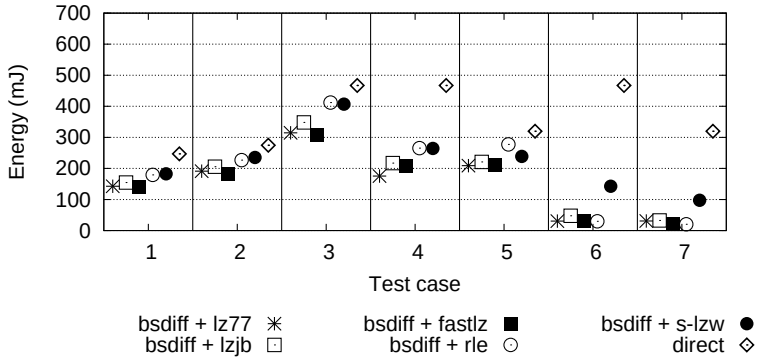
3.4.2.4 Energy estimation

For updating two nodes, using only compressed updates (Figure 3.17c) does not always reduce energy usage when compared with direct transmission of uncompressed binaries. The additional processing pays off in some cases, but it is not significant enough.

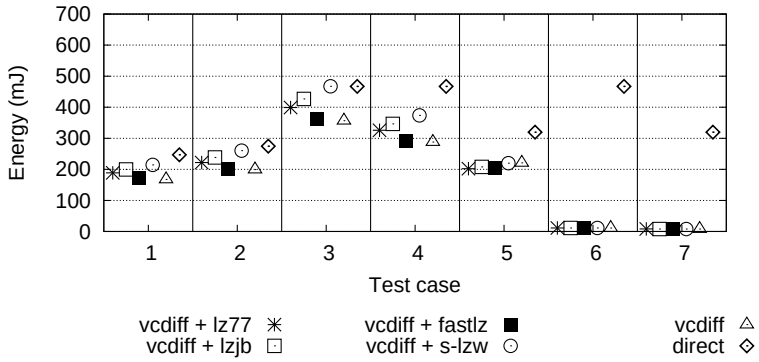
On the other hand, the combination of any compression algorithm with either BSDIFF (Figure 3.17a) or VCDIFF (Figure 3.17b) results in significant reductions in energy usage. For test cases 1 to 5, BSDIFF with LZ77 or FastLZ show highest energy savings, while for test cases 6 and 7, the lowest energy usage is registered using VCDIFF.

VCDIFF has good performance even without using an additional compressor. In fact, only FastLZ reduces the energy usage in all test cases. In the parameter change test cases, since the VCDIFF delta fits in one packet, there is no need to additionally compress it.

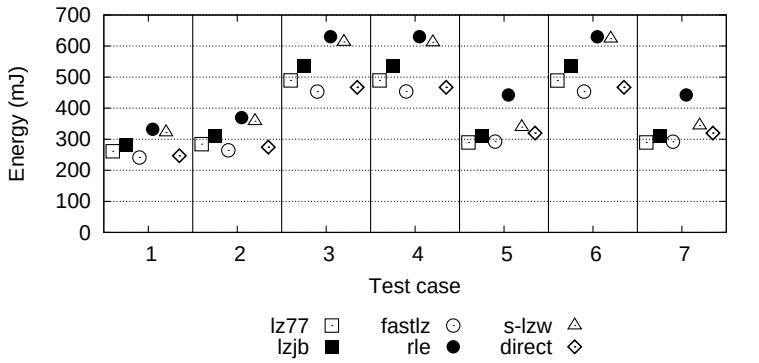
If we take a closer look at two test cases, and we vary the size of the network (Figure 3.18), we can see that the improvements in compression ratio start to overweigh the additional processing time introduced by decompression and patching. As the network size grows, we observe largest benefits when BSDIFF in combination with LZ77 is used, very closely followed by FastLZ. This suggests that the additional processing starts to pay off very soon, even after only three hops.



(a) BSDIFF

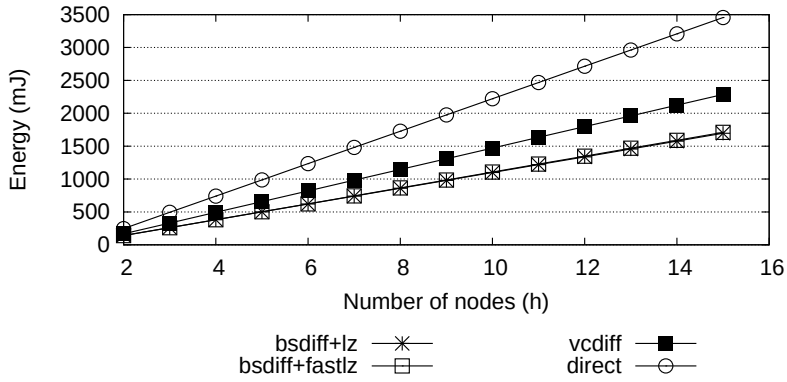


(b) VCDIFF

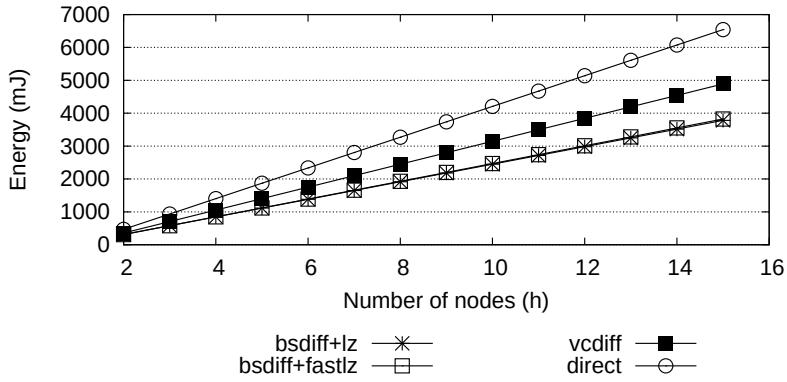


(c) Only decompression

Figure 3.17: Energy estimation using only decompression (c) and both patching and decompression (a, b). (Constants: $h = 2$, $k_{\texttt{err}} = 1$, $payload\_size = 71B$, *buffer size* = 128 B). "Direct" shows the energy usage of transmitting the data directly, without processing.



(a) Test case 1



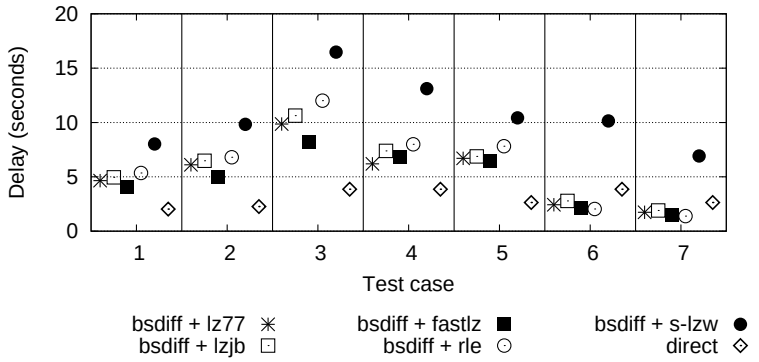
(b) Test case 3

Figure 3.18: Influence of number of nodes (h) on energy consumption.

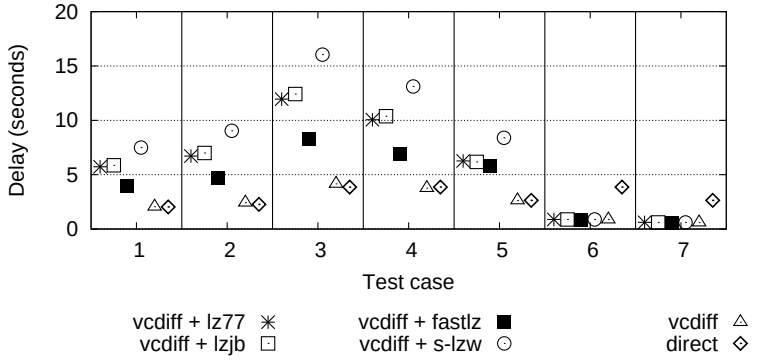
3.4.2.5 Delay

For updating two nodes, using only compressed updates (Figure 3.19c) is much slower than sending the uncompressed binaries directly. Slightly improved results are obtained with incremental updates, as the processing time is still larger than the transmission time. This is evident both for BSDIFF (Figure 3.19a) and VCDIFF (Figure 3.19b) in test cases 1 to 5. Using only VCDIFF is the best option in these cases.

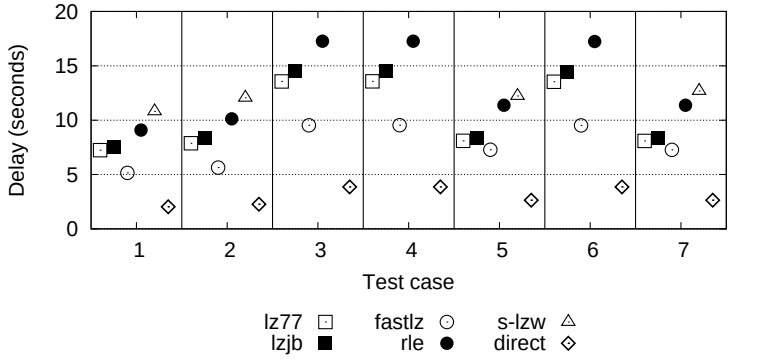
In test cases 6 and 7, the processing overhead is significantly smaller compared to the transmission savings. Therefore, using LZ77, FastLZ or LZJB with BSDIFF, as well as only VCDIFF, is faster than transmitting the entire binary data.



(a) BSDIFF

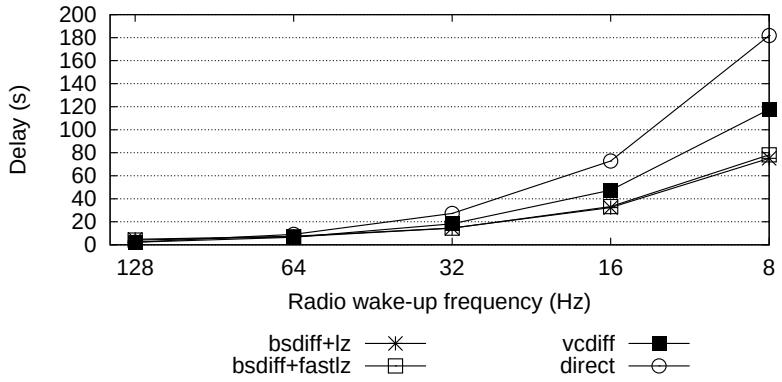


(b) VCDIFF

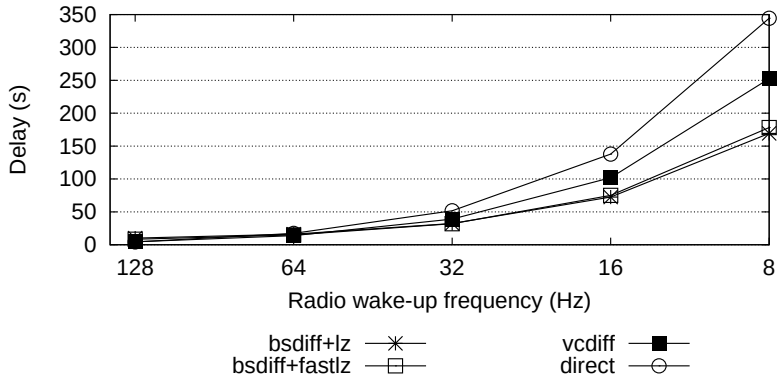


(c) Only decompression

Figure 3.19: Delay estimation using only decompression (c) and both patching and decompression (a, b). (Constants: $h = 2$, $k_{\mathtt{err}} = 1$, $payload\_size = 71B$, *buffer size* = 128 B). "Direct" shows the delay of transmitting the data directly, without processing.



(a) Test case 1



(b) Test case 3

Figure 3.20: Influence of radio duty cycling ($t_{tx/rx}$) on delay.

If we take a closer look at the two test cases as in the previous section, and this time we vary the duration of the transmission time, i.e. we assume that some form of Radio Duty Cycling (RDC) is used (Figure 3.20), we again observe the benefits of smaller updates for transmission. Even in a network of two nodes, with duty cycles of 32 Hz (31.25ms for each transmission), BSDIFF with LZ77/FastLZ finishes faster than VCDIFF. This suggests that factors which influence transmission time, as duty cycling, packet loss, and bandwidth, favor smaller data for transmission.

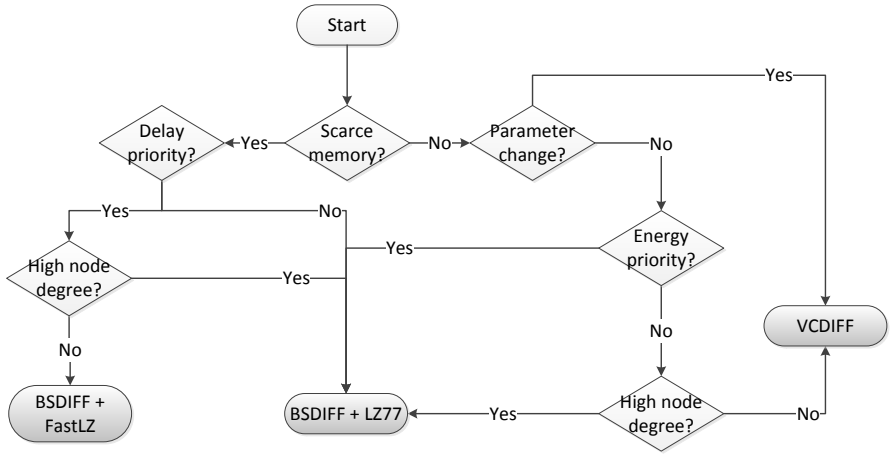


Figure 3.21: Guidelines for selecting the best option for incremental update.

3.4.2.6 Summary

The presented results suggest that reprogramming can be improved in terms of energy efficiency and time required for update by using data compression and incremental updates. Improvements vary depending on the selection of algorithms.

Simply adding compression does not lead to lower energy usage or faster updates. In fact, some compression algorithms can degrade performance.

In contrast, using incremental updates showed solid results in all test cases. Up to 95% in energy savings were registered, along with 95% faster updates. Even though highest improvements were found during parameter reconfiguration, the fact that a 35% reduction in energy consumption was the minimum measured in specific configurations, gives strong arguments for using incremental updates in WSNs.

Selecting the best approach for incremental updates depends on the particular system. The four important factors that influence the selection are available resources, update type, network size and optimization goal (energy or delay). The choice is between using BSDIFF with either LZ77 or FastLZ, or using only VCDIFF. The decision tree, populated by recursively partitioning the gathered results, is shown in Figure 3.21.

3.4.3 Horizontal patching

In this section we evaluate the performance of horizontal patching. First, we describe the experimental setup and the test sets for the experiments. Then, we present the comparison between horizontal and vertical patching, as well as the difference between the greedy approach to horizontal patching and the optimal horizontal delta.

3.4.3.1 Experimental setup

We analyze the performance of horizontal patching when updating the firmware of all devices in a network using a broadcast scheme for distributing updates. We assume that all devices are of different type, and for each device a separate vertical delta is generated. The number of devices in the network depends on the test set being used. The first test set is for updating the firmware of sensor nodes and the second test set is for updating the operating system of smart phones.

Table 3.3: Size of test data (compressed firmware image consisting of an application and an operating system) for sensor nodes, in bytes.

Application	Contiki 2.3	Contiki 2.4	Contiki 2.5
1	25,403	25,563	25,062
2	22,544	21,594	22,579
3	18,324	17,235	18,282
4	17,739	16,696	17,752
5	14,379	13,305	14,490
6	14,027	12,954	14,112
7	14,026	12,941	14,066

We rely on compression ratio, defined in the previous section, as the comparison metric. We use the sum of the compressed new software bundles $\sum_{i=0}^{n-1} C(S_i^{'})$, as a reference point. For each combination of two or more different devices, we compute all vertical deltas. Then between each pair of vertical deltas, we compute all possible horizontal deltas. From these deltas, we find two options for horizontal patching: the optimal combination of vertical and horizontal deltas, and a greedy combination, as explained in the previous section. In the end, we compute the compression ratio of using only vertical deltas, and using the optimal/greedy horizontal delta, using the compressed software

Table 3.4: Size of compressed Android firmware images for Google Nexus devices[1], in megabytes. Devices 1, 4 and 7 form subset 1, for updates from version 4.0.4 to 4.1.1; devices 2, 3, 5 and 6 form subset 2, for updates from version 4.0.4 to 4.1.2 and devices 2, 3 and 8 form subset 3 for updates from version 4.1.2 to 4.2.1.

Number	Device	Android version			
		4.0.4	4.1.1	4.1.2	4.2.1
1	Galaxy Nexus Verizon CDMA/LTE	193	239	-	-
2	Galaxy Nexus (GSM/HSPA+) with Google Wallet	191	-	240	256
3	Galaxy Nexus (GSM/HSPA+)	187	-	234	248
4	Nexus S 4G	169	200	-	-
5	Neux S (Worldwide)	163	-	195	-
6	Nexus S (850 MHz Worldwide)	163	-	195	-
7	Nexus S (Korea)	148	175	-	-
8	Nexus 7 Wi-Fi	-	-	256	272

[1] Available at https://developers.google.com/android/nexus/images

bundles as reference. To prove general applicability, we apply the same process using both BSDIFF and VCDIFF.

The first test set consists of seven applications for the Contiki operating system. They are built together with the operating system into one firmware image for TelosB nodes. We consider three consecutive operating system updates, as shown in Table 3.3. In all test cases, the applications are ordered by size, from largest to smallest. Based on the results from the previous section, we use LZ77 for compressing both vertical and horizontal deltas.

The second test set consists of updates of the Android firmware image of different Google Nexus devices. The devices have different hardware components, such as radio chipsets and sensors. Since vendors rarely maintain the software in such devices for a long time, not all versions of the operating system are available for all devices. Therefore, we split the sample set into three subsets, in which an update from the old and new version exists for each model (Table 3.4). Since Android-based devices are not as resource constrained as sensor nodes, in this test case we use BZip2 [v] as the compression algorithm.

[v] http://www.bzip.org/

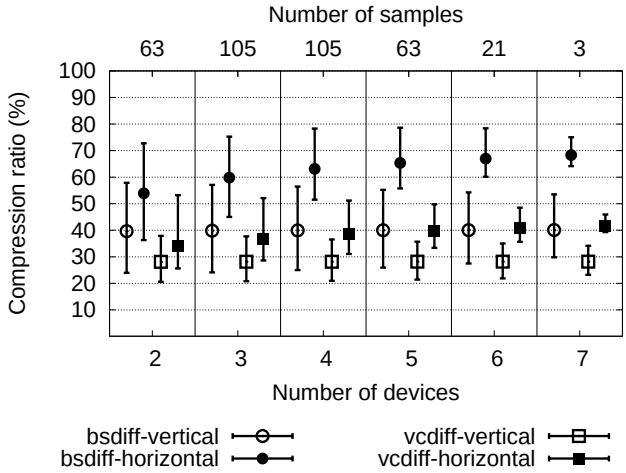


Figure 3.22: Compression ratio of horizontal patching for sensor nodes using BSDIFF and VCDIFF, in comparison to compressed firmware images. Since seven applications in total are available, for three operating system updates, the number of samples available is $\frac{3 \cdot 7!}{k!(7-k)!}, k = 2, 3..7$. BSDIFF has better compression ratio in all cases, although it is considerably slower compared to VCDIFF. In both cases, the compression ratio gained using horizontal patching increases with the number of different devices

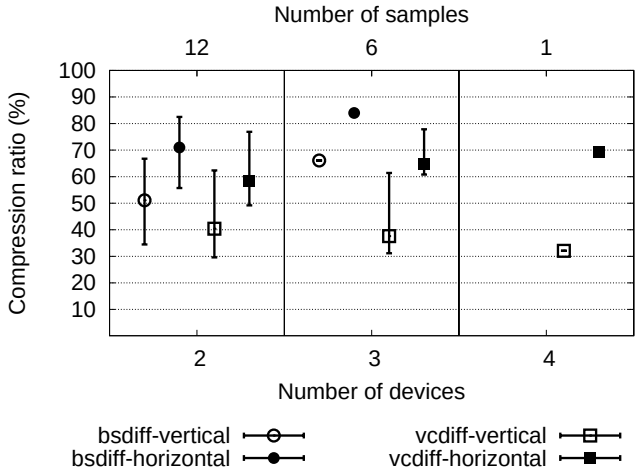


Figure 3.23: Compression ratio of horizontal patching for Android-based devices. It is important to note that we were not able to generate five horizontal patches, therefore the number of samples for BSDIFF for two devices is six, whereas only one sample with three devices was available.

3.4.3.2 Results

Figure 3.22 shows the performance of horizontal patching using the sensor node test set. It is clear that horizontal patching provides higher compression compared to only vertical patching, both for BSDIFF and VCDIFF. The improvement is drastic with BSDIFF. Furthermore, as the number of different devices (applications) rises, the performance of horizontal patching improves, while vertical patching remains stable. For instance, while the average compression ratio of horizontal patching with BSDIFF grows from 56% with two different devices to 71% with seven different devices, the compression ratio of BSDIFF with only vertical deltas is approximately 42% in all cases.

Horizontal patching has similar performance in the second test case (Figure 3.23). Compression ratio is higher with BSDIFF, although both algorithms benefit from horizontal patching compared to only vertical patching. It is important to note that due to the large size of the uncompressed deltas, BSDIFF was unable to produce five horizontal deltas in reasonable time. Therefore, the number of available samples for BSDIFF for this test set is much lower.

Table 3.5: Difference in compression ratio (%) between optimal horizontal deltas and greedy horizontal deltas.

Sample set	Number of devices	BSDIFF		VCDIFF	
		Avgerage difference	Standard deviation	Average difference	Standard deviation
Sensor nodes	2	0.008	0.036	0.014	0.036
	3	0.095	0.169	0.072	0.111
	4	0.137	0.167	0.101	0.112
	5	0.162	0.148	0.120	0.107
	6	0.191	0.118	0.128	0.101
	7	0.232	0.064	0.129	0.087
Smart phones	2	0.001	0.004	0.154	0.316
	3	0.644	-	0.443	0.645
	4	-	-	0.107	

Figures 3.22 and 3.23 show only the best (optimal) horizontal delta. As previously stated, in order to find the optimal delta, horizontal deltas between all possible combinations of pairs of verticals deltas have to be generated, which can consume a lot of time. On the other hand, the greedy algorithm for building the horizontal delta is not far off the

optimal one. As shown in Table 3.5, the difference between the greedy approach and the optimal one is very small, and can be considered negligible when compared with the overall savings of the approach. Therefore, the greedy approach is a sufficient solution to solve the problem of intractability of horizontal patching for large input data and high number of different devices for update.

3.5 Conclusion

In this chapter, we focused on software management in the IoT, addressing research question **RQ3**. We identified that remote software update in LLNs is a slow and tedious task due to the size of data for transmission, imperfect media for transport, and the resource constraints in devices. As a potential solution, we investigated how reduction in data for transmission can help with software updates. We investigated three approaches. Firstly, we evaluated the performance of general purpose data compression algorithms applied directly on binary data. Secondly, we compared three algorithms for incremental update and combined them with the previously analyzed data compression algorithms. Finally, we presented horizontal patching, a method for optimizing the size of incremental updates in a multi-application environment where heterogeneous devices share a common software component. Horizontal patching reduces the size of updates by constructing one delta from another. Therefore, when the common software component needs to be updated, horizontal patching can be used to create a smaller delta compared to the collection of deltas for each individual device.

Results show that data compression in combination with incremental updates can significantly decrease energy usage and delay in reprogramming WSNs, but a bad choice can also increase it. The best option to perform incremental updates depends on multiple factors, for which we have provided a decision tree. Best performance was measured when using either the VCDIFF delta encoding algorithm, or the combination of BSDIFF for delta encoding and LZ77 or FastLZ for decompression.

Horizontal patching gives better results as the number of heterogeneous devices for update grows. This comes at the cost of additional processing time required for computing all horizontal deltas. The scalability analysis shows that the number of possible options for horizontal patching quickly grows and it becomes impossible to compute the final outcome.

Therefore, a greedy approach is presented, which only searches through horizontal deltas which have a root in the largest vertical delta.

The improvement of horizontal patching is confirmed by experimental validation for updating software in both resource constrained devices and modern Android-based devices. The impact is evident with two algorithms for delta encoding – BSDIFF and VCDIFF. For instance, with BSDIFF, the average compression ratio of vertical patching is around 42% for two to seven different devices in the first test set, while the compression ratio of horizontal patching grows from 56% with two different devices to 71% with seven different devices. Similar results are measured with VCDIFF, with 31% compression ratio of vertical patching with two to seven different devices, and 39% to 48% compression ratio with two to seven different devices.

In all test cases, the greedy approach is shown to be very close to the optimal horizontal delta. The difference between the optimal and greedy horizontal delta is at most 1.5%, which shows that horizontal patching can be used even with a large number of different devices.

The method of horizontal patching can be easily adopted for updating any devices which share software components. This leads to more efficient schemes for telecom operators to upgrade fleets of smart phones, tablets, television sets, and other consumer devices, using smaller updates.

4

Service Discovery

In this chapter, we investigate how powerful and resource constrained devices can be integrated in a single logical network (research question RQ2). As a potential solution, we look at integration at the service discovery level: the adoption of a common service discovery protocol used by all devices, which enables them to autonomously, at run-time, discover each other's existence, their capabilities and available services. After an initial survey, we select the standardized Multicast DNS with DNS-Based Service Discovery (mDNS/DNS-SD) protocol as a viable candidate, and analyse its applicability in the Internet of Things (IoT) (research question RQ4).

We show that mDNS/DNS-SD requires devices to be always online, which is inappropriate for battery powered devices, and its discovery features are not discriminative enough to be used in large networks. As a solution, we propose a proxy scheme, where resource constrained devices delegate their service discovery responsibilities to proxy servers. Then, we describe a new naming scheme, which enables devices to be looked up based on their physical properties, such as location and available sensors. Both extensions are backward-compatible with the mDNS/DNS-SD standard, making them favourable for use in the IoT.

The chapter is based on the following publications:

- Milosh Stolikj, Johan J. Lukkien, Pieter J. L. Cuijpers, and Nina Buchina. "Nomadic Service Discovery in Smart Cities". In: *Smart Cities and Homes: Key Enabling Technologies*. Ed. by Mohammad S. Obaidat and Petros Nicopolitidis. Elsevier, 2015.
- Milosh Stolikj, Richard Verhoeven, Pieter J.L. Cuijpers, and Johan J. Lukkien. "Proxy support for service discovery using mDNS/DNS-SD in low power networks". In: *IEEE Symposium on A World of Wireless, Mobile and Multimedia Networks*. WoWMoM. 2014, pp. 1–6.

4.1 Introduction

Service discovery is a key component of distributed systems. It is the process of finding services offered by service providers within a given network, that match the requirements of the service seeker (called *client* from now on). Protocols for service discovery are of particular importance for self-organizing systems, such as large Machine to Machine (M2M) networks in the IoT, with many service clients and potential service providers. In these networks, service discovery protocols are the principal component which allows devices to explore their environment. These service discovery protocols are expected to operate in autonomous fashion, without support from an end user. Furthermore, they should handle changing networking conditions and temporarily unavailable service providers. Finally, due to the scale of the network, the communication load should be kept as low as possible.

Figure 4.1 highlights the process and the goal of service discovery: service seekers issue queries matched by service providers in a distributed context where neither one knows the existence of the other. Service discovery protocols achieve this goal by defining 1) a common language for describing services and selection criteria; 2) a common protocol for exchanging service descriptions between service clients and service providers; and 3) rules for matching service descriptions with the selection criteria.

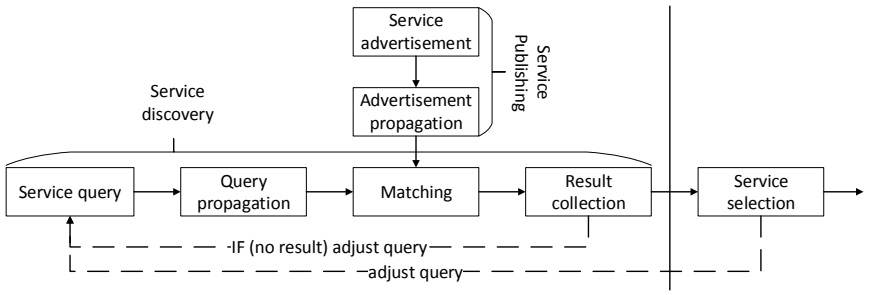


Figure 4.1: The service discovery process. The service seeker issues a query that is matched by the service advertisement of the service provider. In a distributed context, these tasks can be performed by different entities. Figure taken from [SLCB15].

In this chapter, we aim to identify a suitable protocol for service discovery in the IoT. This protocol must therefore work for very constrained devices

and networks. An important requirement is that it follows the Internet Engineering Task Force (IETF) standardization procedures, in order to guarantee wide acceptance. From the plethora of protocols, we focus on the Multicast Domain Name System (mDNS) [RFC6762] with DNS-Based Service Discovery (DNS-SD) [RFC6763], a standards-based protocol that already partially satisfies requirements **R1**, **R3**, **R4** and **R5** from Section 2.3.5. The protocol has good scalability, light footprint, and wide usage.

DNS-SD is an extension of the Domain Name System (DNS) which uses DNS Resource Records (RRs) to describe services in a domain-name like fashion. The service description can further specify the service protocol and additional context. Similarly, mDNS is an extension of the DNS protocol for resolution in Local Area Networks (LANs). The main differences between mDNS and the original DNS protocols are: 1) naming information is stored locally, on each node in the network; 2) resolution queries are sent multicast instead of unicast; and 3) each node directly responds to queries.

The mDNS/DNS-SD protocol in its current form contains several issues, which hinder its performance in Low-Power and Lossy Networks (LLNs). Firstly, mDNS/DNS-SD assumes service providers to be constantly online, which is impossible for battery operated devices. Even though this issue can be amended by complementary protocols, such as network-wide caching, such approaches are not standardized. Secondly, the services are described using very descriptive, long names. As a result, service descriptions are relatively large, and require packet fragmentation when used over LLNs with small payloads, as IEEE 802.15.4. Finally, the protocol is limited in the query language, and it facilitates only coarse selection criteria in terms of service type. In networks with many similar devices, as in the IoT, this increases the traffic load.

In this chapter, we propose solutions for the stated three issues with mDNS/DNS-SD.

First, we present an extension to the mDNS/DNS-SD protocol which adds proxy servers. Proxy servers take over service discovery responsibilities from battery-operated service providers. This reduces traffic load, and enables battery-operated service providers to be unavailable for longer periods of time, yet still remain discoverable (Requirement **R2**).

Second, we extend DNS-SD with a context model for service descriptions. We enable services to be associated with context tags, and service

clients can query for services with predicates formed by context tags and boolean operators. This extension improves the scalability of the protocol in large networks. (Requirement **R1**).

Third, we propose a compression method to avoid long descriptive names inside packets. With the proposed change, service descriptions can be stored in a more compact form, and can be delivered using smaller payloads without fragmentation (Requirements **R1** and **R2**).

The chapter is structured as follows. In Section 4.2 we cover related work on service discovery. In Section 4.3 we describe the mDNS/DNS-SD protocol and identify its key issues when used in large networks. We present the protocol extensions for proxy support in Section 4.4, and the context tag model in Section 4.5. Finally, we present the proposed compression method in Section 4.6 and give concluding remarks in Section 4.7.

4.1.1 Background

The main objective of service discovery protocols is to find a service provider that satisfies some given criteria. However, this is a complex task, which consists of several subtasks. First, the service discovery protocol needs to have a language for specifying services and selection criteria. The language also defines how a given service description matches a given selection criteria. Then, the service discovery protocol defines how and where the service descriptions are stored, and how they are accessed.

The main parties involved in service discovery are the service client and the service provider. The service client provides the selection criteria for service discovery, while the service provider hosts a specific service, called a *service instance*. Service discovery protocols often use an intermediate party for storing and accessing service descriptions, commonly referred to as a service repository, directory or broker.

Several studies have focused on identifying the key properties of service discovery protocols for LLNs [MHS05; ZMN05]. Based on these studies, we select the following features to characterize service discovery protocols for the IoT: architecture, message scope during service discovery and/or service advertisement/registration, service description language, overhead traffic and inter-operability with existing protocols. As additional features we also consider awareness of mobile service providers,

caching support, detailed service descriptions and energy and resource demands.

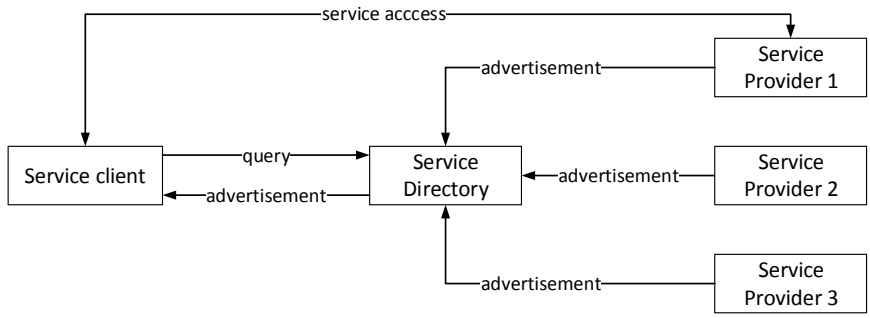


Figure 4.2: Centralized service discovery. The service clients issues a query to a service directory, which contains a list of all known services.

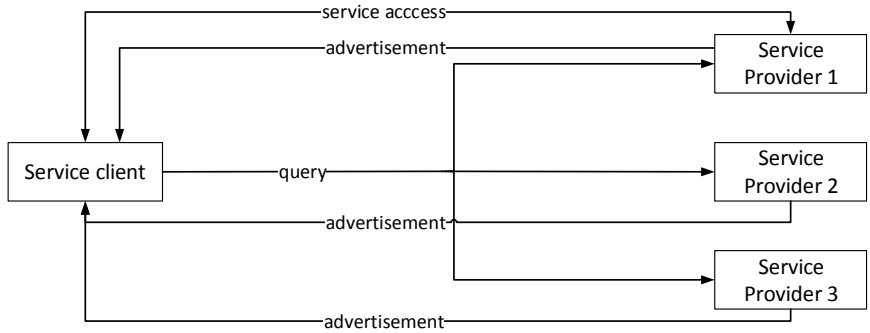


Figure 4.3: Distributed service discovery. The service clients issues a query to all service providers, and each of them responds directly to the client.

The communication patterns between the three parties defines the architecture of service discovery protocols. We distinguish three architectures: centralized, distributed and hierarchical. In a centralized architecture (Figure 4.2), all service descriptions are stored in one or more service repositories. Service providers register their services in the service repository, and service clients look for services in service repositories. In a distributed architecture (Figure 4.3), service descriptions are stored locally on each service provider. Therefore, service clients need to inquire all service providers whether they satisfy the selection criteria. Hierarchical architectures are a combination of both previous architectures (Figure 4.4). They organize service providers in subgroups or clusters, and elect cluster heads as service repositories for the given group. There-

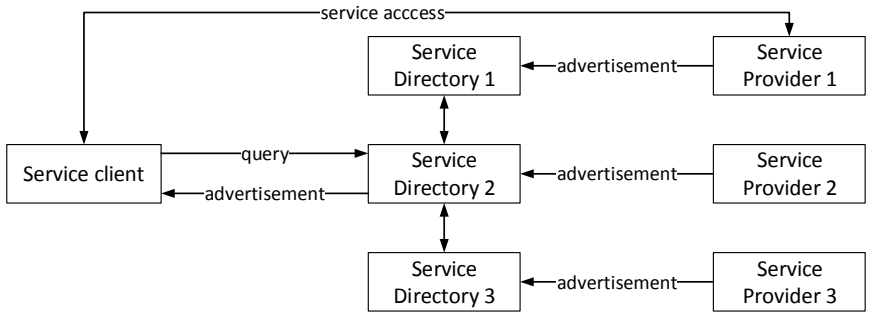


Figure 4.4: Hierarchical service discovery. The service clients issues a query to a service directory, which either responds with an entry from its own list, or forwards the query to other directories.

fore, service providers register services with cluster heads, and service clients contact only cluster heads during service discovery.

Each of the three architectures has benefits and drawbacks. Centralized service discovery protocols take most of the workload concerning service discovery from service providers, as they only need to initially register services, and service clients need to inspect only a single service repository. However, centralized service discovery protocols require a more powerful service repository, which has to be pre-configured in the entire network, and represents a single point of failure. Distributed service discovery protocols require more complex actions from all nodes participating in the service discovery protocol, which can influence their lifetime. Furthermore, messages exchanged are usually broadcast based, and may flood the network. Finally, hierarchical architectures depend on clustering algorithms for creating the initial clusters and selecting appropriate cluster heads. While communication is limited within clusters, additional overhead is introduced while clusters are being established and cluster heads are elected.

4.2 Related work

Previous research in service description and service discovery protocols for LLNs can be roughly categorized into two groups: general service discovery protocols and their application in LLNs, and custom service discovery protocols for LLNs. Naturally, service discovery protocols for

LLNs are considerably different from service discovery protocols for high capacity networks. Therefore, we will first cover related work on general-purpose service discovery protocols, before describing custom service discovery protocols for LLNs. The main characteristics of all covered protocols are summarized in Tables 4.1 and 4.2. Finally, we will discuss the applicability of existing protocols in the IoT domain.

4.2.1 General-purpose SD protocols

Many service discovery protocols have been proposed since the adoption of Service Oriented Architecture (SOA) as a useful paradigm. Among them, only a few protocols have found wider acceptance, and have been standardized. Recently, some of them have been considered for adaptation for LLNs as well. We now given an overview of the most important ones, with emphasis on portability and scalability.

mDNS/DNS-SD is a widely used standard for service discovery in LANs. It uses a distributed architecture, with services described as domain names. mDNS/DNS-SD has been implemented in all major operating systems for high capacity devices, as well as in resource constrained devices [KK12b; STS11]. Further optimizations of the protocol have been described in [KK13], which reduce message sizes in order to support LLNs better. We describe the protocol in more detail in Section 4.3.

Java Intelligent Network Interface (JINI) [Wal00] is a centralized service discovery protocol for the Java programming environment. JINI uses Java interfaces for describing services and Remote Method Invocation for accessing them. Service descriptions are stored in lookup servers, which are discovered using multicast messages. Due to the memory requirements of the Java virtual machine, JINI is rarely used in LLNs.

Simple Service Discovery Protocol (SSDP) [Pre+08] is part of the Universal Plug and Play (UPnP) specification, a commonly used software stack for consumer electronics devices. It is a hierarchical protocol, where participating entities register and resolve services using control points. The messaging protocol is based on the Hyper Text Transfer Protocol (HTTP) standard, with multicast messages for advertising and discovering new services, and unicast responses. Services are described as Uniform Resource Locators (URLs) for control and eventing. The overhead brought by the transport protocol used in SSDP is unsuitable for

LLNs. Therefore, most efforts for interconnecting consumer electronics devices using SSDP with LLNs have relied on some form of gateways to translate messages between the two networks [BLV11; SCNFR11].

Service Location Protocol (SLP) [RFC2608] is another standardized service discovery protocol for LANs. SLP has been designed to scale from small, decentralized networks to large corporate networks, by supporting fully distributed or centralized operation. SLP defines three types of entities: User Agents (UAs) which request services, Service Agents (SAs) which provide services and Directory Agents (DAs) which cache service advertisements. DAs are optional, but if they exist in a network, both SAs and UAs are obliged to use them. The proposed IPv6 over LoWPAN (6LoWPAN) adaptation of the protocol called Simple Service Location Protocol (SSLP) [Kim+10] introduces a new Translation Agent (TA) which translates messages between a 6LoWPAN and a high capacity network running SSLP. DAs are used as caching entities, which limit the scope of advertisements and queries. In both protocols, discovery and advertisements messages are multicast if DAs are not used, and responses are always unicast. Unfortunately, the 6LoWPAN adaptation of the SSLP protocol seems abandoned. Furthermore, SLP has been extended with proxy agents [CJAK06], for connecting high and low power networks, and context support such as proximity services [Cha+08].

4.2.2 SD protocols for LLNs

Many service discovery protocols have been designed with resource constrained devices in mind. These protocols can operate in a single-hop network [Blu12; Nid01], clusters of nodes [SBR04; MPSHH06], centralized environments [OLBU10], or 6LoWPAN networks [ARYK10; RAYK10; BPGO12]. We now give an overview of some of the most popular service discovery protocols for resource constrained devices.

Service Discovery Protocol (SDP) [Blu12] is part of the Bluetooth standard for locating available server applications, and learning about their characteristics in one-hop ad-hoc networks. Services are resolved through SDP servers, which run on every Bluetooth device offering services. The protocol itself does not specify how SDP servers are selected by clients, or how they detect when they become unavailable.

DEAPSpace [Nid01] is a proactive distributed service discovery protocol for use in single-hop networks. It uses broadcast messages to advertise

all known services to neighbouring devices at regular intervals. Since each device advertises services offered by other neighbouring devices, low powered nodes may choose to have low advertisement intervals. The algorithm does not scale in large networks, due to 1) broadcast storms in multi-hop environments with large number of nodes and 2) large size of advertisements when many service providers coexist.

Konark [HDVL03] is a distributed service discovery and service delivery protocol for ad-hoc networks. It uses multicast messages for both periodic service advertisements and service discovery queries. Services are described in Extensible Markup Language (XML), similar to the Web Services Description Language (WSDL) and delivered in the form of URLs. Konark has been designed and tested for ad-hoc networks of high capacity devices, and the underlying protocols used, such as HTTP, are not suitable for LLNs.

SANDMAN [SBR04] is an energy-aware hierarchical service discovery protocol. It assumes that an arbitrary set of mobile devices within a network have similar mobility patterns, and can therefore be grouped in a cluster. Then, from the cluster, only one device answers to service discovery queries, while the others can sleep. SANDMAN does not include a cluster management protocol, and it is therefore difficult to estimate its general applicability.

Sleeper [BBCF06] is a distributed service discovery protocol, using proxies for delegating answers to service discovery queries. This way, constrained nodes can endure long sleeping times while other nodes can still discover their services. Sleeper supports various description options for services, including geographic location, meta data and ontology information. Service descriptions are enriched with popularity metrics, which are used by proxy nodes to select which advertisements will be cached. Services can be discovered proactively or reactively, i.e. using periodic service advertisements or explicit service discovery queries. Sleeper does not cover the election process for proxy nodes. The protocol is evaluated on high capacity devices using IEEE 802.11b, and it is therefore unknown how it performs in LLNs.

Open Service Architecture for Sensors (OSAS) uses a centralized approach for service discovery [OLBU10]. It relies on a Resource Manager (RM) for holding service advertisements, which are populated from constrained nodes at regular intervals. Since there is one resource manager, unicast messages are used at all times except when discovering

new nodes. A centralized solution is useful for small networks, such as body sensor networks or smart office spaces, but does not scale to large numbers. Introducing multiple resource managers would make this protocol similar to the other clustering protocols.

The Electronic Number Mapping (ENUM) service discovery protocol [ARYK10] has been proposed as a possible solution for service discovery in 6LoWPAN networks. It uses compressed entries for describing service endpoints, which are then resolved using standard DNS queries. The architecture is hierarchical and assumed to be pre-configured, with sensor nodes connected to master nodes, which are globally addressable and interconnected with gateway nodes. The protocol uses unicast messages between sensor nodes and master nodes, but does not specify how master nodes are detected.

Fast and Energy Efficient Service Provisioning (FESP) [RAYK10]) is a protocol for managing already discovered, frequently used services in 6LoWPANs. It assumes that initial discovery has already been carried out by another protocol. Afterwards, it improves service discovery latency by 1) sending service discovery queries only to immediate, 2-hop neighbours; and 2) as last resort, sending queries to a local gateway. While such metrics can be seen as optimizations to other service discovery protocols, relying on every node to cache all popular service descriptions is inappropriate in constrained nodes.

TRENDY [BPGO12] is a centralized context-aware service discovery protocol for 6LoWPANs. Similar to SLP, it has one central directory agent, but other SAs and UAs are organized in clusters, with Group Leaders (GLs) on top. Unlike cluster heads, GLs are not elected by surrounding nodes, nor store service advertisements. They are designated by the DAs, and merely monitor the status of SAs. The entire protocol is built on top of the Constrained Application Protocol (CoAP) [SHB13] and services are described as Uniform Resource Identifiers (URIs) with variable parameters, including location and type. Caching and proxy support are taken from the CoAP base. Certain information like the DA address has to be either hardcoded or discovered through another protocol.

4.2.3 Summary: Solution for the IoT

Obviously, the service discovery research field is very fragmented, with various protocols under development or in use. Such heterogeneity is unsuitable for the IoT, as an intermediate translation agent would be required between any set of devices which use a different service discovery protocol. Therefore, we believe that accepting a single solution for an service discovery protocol would be beneficial for easier integration in the IoT.

Unfortunately, none of the existing protocols are ideal. They either lack in features, are too specific for certain tasks, or simply have not been accepted in the community. In [But14], based on the availability, expressiveness, and resource requirements, three of the described protocols have been identified as suitable candidates: SSLP, TRENDY and mDNS/DNS-SD. SSLP was was discarded because it requires application-level gateways for backward compatibility, lack of support for sleepy nodes, and large overhead. These issues are amended in TRENDY, which is proposed as the best protocol.

However, TRENDY acts as an extension of the CoAP protocol, which imposes requirements at the application layer. Even though CoAP is used heavily in the IoT at the moment, other application protocols in use, as Message Queuing Telemetry Transport (MQTT) and Extensible Messaging and Presence Protocol (XMPP), cannot be discarded. When a different application protocol is in use, implementing a CoAP-based service discovery protocol would be an additional burden. According to [But14], TRENDY with a limited CoAP implementation requires 9.34kB of ROM memory and 0.86kB of RAM memory, plus an additional overhead for any services for advertisement. Finally, since TRENDY is not standardized yet, there has been no wider acceptance of the protocol.

In the same study, mDNS/DNS-SD was discarded due to lack of context-awareness and heavy traffic. However, recent activities in the IETF standardization body are focused towards the development of a new version of the protocol, viable for IoT usage. The work presented in this chapter is in line with these efforts, by tackling the aforementioned weaknesses of mDNS/DNS-SD.

Table 4.1: Comparison of service discovery protocols

Protocol	Architecture	Discovery mechanism	Registration mechanism	Overhead	Description	Interoperability
mDNS/DNS-SD [RFC6762; RFC6763]	distributed	multicast	multicast	DNS packets	DNS-SD	yes
Jini [Wal00]	centralized	multicast or unicast	multicast or unicast	heavy protocol	Java	yes
UPnP [Pre+08]	hierarchical	multicast or unicast	multicast	heavy transport protocol	URL	yes, gateways
SLP [RFC2608]	distributed or cetralized	multicast or unicast	multicast or unicast	translation agents	string, XMLS	yes
SDP [Blu12]	distributed	unicast	unicast	SDP server discovery (unknown)	UUID	no
DEAPSpace [Nid01]	distributed	broadcast	proactive	periodic advertisements	none	no
Konark [HDVL03]	distributed	multicast	multicast	periodic advertisements	XML	no
SANDMAN [SBR04]	hierarchical	unicast	unicast	cluster maintenance, sleep announcements	none	no
Sleeper [BBCF06]	distributed	broadcast	broadcast	proxy election, periodic advertisements	taxonomy	no
OSAS [OLBU10]	centralized	unicast	unicast	periodic advertisements (heartbeat)	none	no
Anwar et al [ARYK10]	hierarchical	unicast	unicast	cluster maintenance	ENUM/DNS	DNS
TRENDY [BPGO12]	centralized or hierarchical	multicast	multicast	periodic updates, group leader election	CoRE, URI	no

Table 4.2: Comparison of service discovery protocols

Protocol	Localization	Mobility	Proxy	Energy-aware	Low resource
mDNS/DNS-SD [RFC6762; RFC6763]	through domain name	time-to-live expiry, service invalidation	yes	no	yes [KK12b; STS11]
Jini [Wal00]	no	timeout expiry	yes	no	no
UPnP [Pre+08]	through domain name	timeout expiry, periodic advertisements	no	no	yes
SLP [RFC2608]	yes [Cha+08]	timeout expiry (check)	yes	no	yes [CJAK06]
SDP [Blu12]	yes (one hop)	no	no	yes	yes
DEAPSpace [Nid01]	yes (one hop)	periodic advertisements	yes	yes	yes
Konark [HDVL03]	yes, in description	timeout expiry, periodic advertisements	no	yes	unknown
SANDMAN [SBR04]	yes, in clusters	moving clusters	cluster heads	yes	unknown
Sleeper [BBCF06]	yes, in description	timeout expiry, service invalidation	yes	yes	no
OSAS [OLBU10]	no	timeout expiry	yes	yes	yes
Anwar et al [ARYK10]	yes, in identifier	timeout expiry	yes	no	unknown
TRENDY [BPGO12]	yes, description	periodic advertisements	yes	yes	yes

4.3 mDNS/DNS-SD service discovery

mDNS and DNS-SD form the core service discovery protocol in Bonjour, a Zero-configuration implementation by Apple. The protocol consists of two components: a communication protocol defined by mDNS, and a service discovery and service description protocol defined by DNS-SD.

mDNS is an extension over the unicast DNS protocol [RFC1034; RFC1035] for name resolution. Names can refer to addresses, as in classic DNS, or to services, using DNS-SD. The primary purpose of mDNS is to enhance name resolution in a LAN. Therefore, mDNS exclusively resolves host names ending with the *.local* top level domain. The packet structure in mDNS is similar to the one defined in the standard DNS protocol. The main differences between the two comes in the message exchange protocols and in the distribution of resolution information: DNS assumes hierarchically organized servers, which hold all knowledge about the network, and unicast messaging between clients and servers. Contrarily, mDNS foresees a fully distributed environment, where each device directly answers to name resolution queries for its own local entries. In that sense, every participating device acts both as a server and a client. Furthermore, the resolution information is distributed throughout the entire network, with each participating device holding a small subset of it. mDNS uses multicast messaging to efficiently distribute both queries and responses to all devices within the network. mDNS packets are sent to/from the reserved multicast addresses 224.0.0.251 (Internet Protocol version 4 (IPv4)) and ff02::fb (Internet Protocol version 6 (IPv6)), using User Datagram Protocol (UDP) port number 5353.

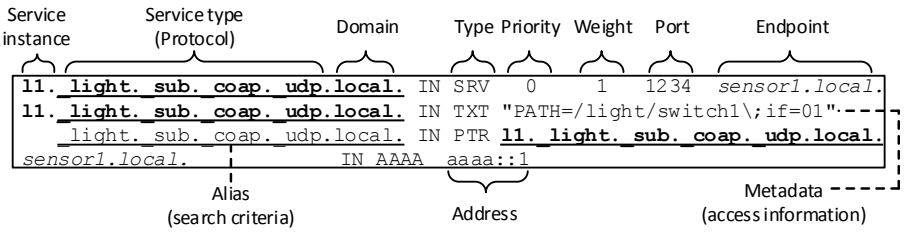


Figure 4.5: DNS-SD description of a light sensor service. The four resource records are connected through the name of the service.

DNS-SD is a standardized protocol for describing and resolving services using DNS RRs. DNS-SD defines how a service client can leverage

standard DNS queries to discover service instances within a logical domain using the service type as selection criteria. DNS-SD describes service instances using SRV, TXT, PTR and A/AAAA RRs (Figure 4.5). The SRV and TXT RRs have the same structured name, in the form "<Name>.<Type>.<Domain>". The first part of the name is a unique identifier of the service instance. The service type is formed by concatenating the application protocol and transport protocol used for accessing the service instance. Lastly, the domain defines the scope of the service instance.

SRV RRs, besides the name of the service instance, contain the port number for accessing the service instance, the priority and weight parameters to discriminate between service instances of the same type, and the host name of the service provider where the service instance is stored. The host name can be resolved to an IPv4/IPv6 address using A/AAAA RRs.

TXT RRs contain additional service metadata in the form of *[key]:[value]* pairs. The exact content depends on the protocol used, and can include an URI path for a specific resource, invocation parameters, and other more specific service descriptions. The maximum size of the TXT RR is 1300 bytes.

PTR RRs have a name in the form '<Type>.<Domain>'. DNS-SD uses PTR RRs to provide a mapping between a service type and a specific service instance. As explained in the following section, they are the key RRs used by clients to discover services.

DNS-SD can be used with the existing DNS infrastructure, or in combination with mDNS. When used with the existing, unicast DNS infrastructure, it enables service discovery within existing DNS scopes. However, in combination with mDNS, it enables plug and play functionality of services within a local broadcast domain. Since there is no need to configure separate DNS servers, auto-configuration is easier. Furthermore, adding new devices to the network is trivial, since the new devices can discover and advertise network services independently.

4.3.1 Operational modes

A client discovers a service instance using mDNS/DNS-SD, by retrieving the four RRs associated with the given service instance. DNS-SD can be used in two modes to enable service discovery: proactive and reactive. In

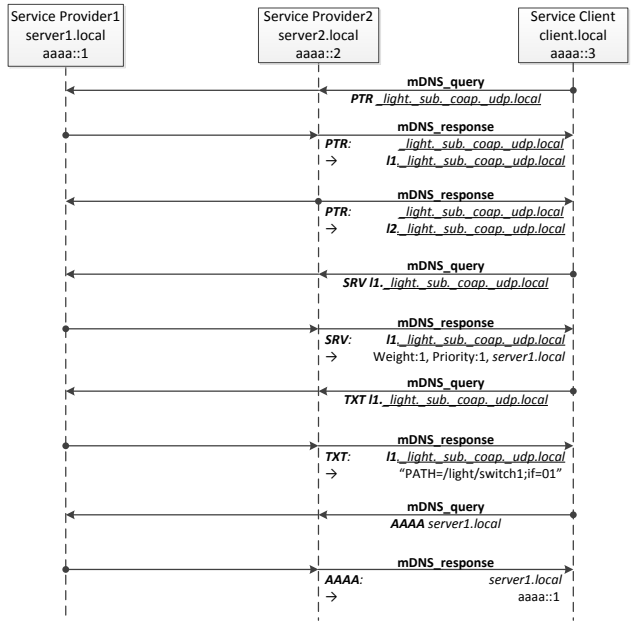
proactive mode, service providers periodically advertise hosted services, by multicasting their descriptions to the network. A service client then needs to listen for these advertisements, and match them against the selection criteria. Obviously, in this mode, there is a large trade-off between overhead traffic and speed of discovery. In large networks, with many service instances, this approach is inappropriate.

In reactive mode, the service client initiates the discovery process by multicasting a query with the selection criteria. The query consists of one or more PTR RRs, with the wanted service types as name. Then, upon receipt, each service provider checks whether locally it has a PTR RR with the given name, and if so, sends it back. mDNS specifies that the response can be either unicast or multicast. Furthermore, the service provider can also include additional RRs in the response, in case the service client asks for them later. If only the PTR RRs is returned, then the service client has to additionally query for SRV, TXT and A/AAAA RRs. Both options are illustrated in Figure 4.6.

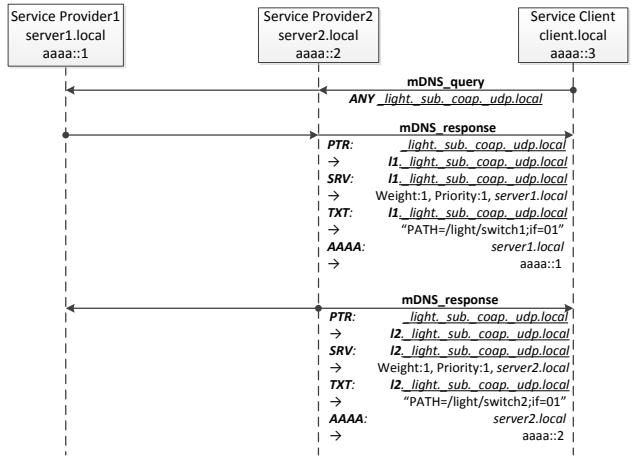
The flexibility of mDNS/DNS-SD imposes some optimization problems: whether responses to queries should be unicast or multicast, and whether they should include complete descriptions (i.e. send all RRs associated with a service instance), or only concrete answers to queries (i.e. send only the matched RRs). An additional complexity is the maximum available payload by lower layers. For example, using DNS name compression, four RRs for one service description occupy around 158 bytes. If IEEE 802.15.4 is used, this DNS packet would be fragmented in at least two frames. We investigate these trade-offs in the next section.

4.3.2 Strategies for responding to queries

In order to compare the different strategies when responding to queries in mDNS/DNS-SD, we develop an analytical model. Consider a network of n nodes, connected in an ad-hoc fashion. In the set of nodes, there is one service client, which sends a single query, and a set of service providers $P = \{p_j, 0 \leq j \leq k, k \leq n\}$, that match the query. A description of a service consists of $r \geq 1$ RRs, where each RR fits in one frame. If all s RRs are bundled in one DNS packet and compressed using DNS name compression, the resulting packet is fragmented in $f \geq 1$ frames. To abstract from the propagation method and the network topology, we assume that the number of frames generated in the network for a single unicast/multicast frame is U/M respectively.



(a) Service discovery without packet fragmentation



(b) Service discovery with packet fragmentation

Figure 4.6: Resolving a service using mDNS/DNS-SD. First, the resolver needs to find service instances of the requested service, provided by PTR RR. Then, the actual service is resolved, through SRV and TXT RRs. Finally, the host providing the service is resolved via A/AAAA RRs. Depending on the implementation, the four RRs can be distributed independently, in separate packets (a), or can be packed into one larger packet (b). Note: all messages are multicast and all nodes belong to the same broadcast domain.

Therefore, in the first strategy, the service client first sends a multicast query, for which all service providers in P respond. Then, the service client selects one of them, and further resolves it, using $r - 1$ queries. The total number of frames in the network is

$$C_s = r \cdot M + k \cdot R + (r - 1) \cdot R, \tag{4.1}$$

where the response R can be either M or U.

In the second strategy, upon the transmission of the initial query, all service providers immediately respond with all RRs for a given service. Then, the total number of frames in the network is

$$C_f = M + k \cdot f \cdot R. \tag{4.2}$$

If we combine these two formulas, we get:

$$C_f \leq C_s \iff k \leq \frac{r-1}{f-1} \cdot (\frac{M}{R} + 1) \tag{4.3}$$

The given equation shows the optimal strategy for responding to queries depends on the number of responses, the effect of the compression method, and weight of the different messaging implementations. We discuss these trade-offs in the following sections.

4.3.2.1 Multicast vs unicast responses

In both strategies, the decision whether to to use multicast or unicast responses depends on the ratio M/U. However, their relationship is more delicate than it seems. At first thought, unicast responses should requires fewer or as many frames for delivery in the network as multicast responses. However, an additional factor to consider is the cost of discovering a route between the service provider and the service client. If this route has been established beforehand, then, since multicast transmissions have a wider scope, $M \geq U$, and sending a unicast response is preferable. However, if such a route is not known, discovering it is commonly done with an additional multicast query, and a unicast response with the route [RFC6550]. As a result, using multicast responses would create less traffic in the network than unicast responses ($M < U$).

In LLNs, a route is usually established and maintained only towards the sinks in the network. Therefore, if the service client is a sink, unicast responses are recommended. However, if other, non-sink nodes in the network are service clients, then maintaining routes towards all of them might be unfeasible. As a result, multicast responses are preferable.

4.3.2.2 Short vs complete responses

Let us assume that the optimal response strategy is used, i.e. $R = min(M, U)$. Since $\frac{M}{U} \geq 1$, from equation 4.3 we can conclude that if $f = 1$, then sending complete responses is always optimal. However, if $f > 1$, the optimal responding strategy depends on the ratio $\frac{M}{U}$ and $\frac{r}{f}$, as in equation 4.3.

Typically, $r = 4$ and $f = 2$ (Figure 4.5). Then, with multicast responses, sending all RRs generates less traffic than single RRs if $k \leq 6$, i.e. there are less than six service providers that match the service client's query. Similarly, when unicast responses are used in large networks, where multicast forwarding generates a lot more traffic than unicast, complete answers are again preferable.

4.3.3 Problems in mDNS/DNS-SD for IoT

Since mDNS/DNS-SD had been designed for service discovery in LANs of high capacity devices, it is not directly applicable to networks of resource constrained devices. We identify the following problems:

1. *Code size*. Due to memory limitations alone, it is impossible to reuse existing mDNS/DNS-SD implementations (Requirement R5). Table 4.3 shows the memory profile of several popular implementations. From the table, it is clear that the original Bonjour implementation cannot fit on resource constrained devices. Even the Arduino port, which is already a feature-limited implementation of mDNS/DNS-SD, is too large to fit on small factor devices.

2. *Energy consumption*. As shown in the previous section, the mDNS/DNS-SD protocol heavily uses multicast messaging. As a result, almost all messages in the network reach all nodes, which consumes significant energy (Requirement R1). Furthermore, service providers need to be online when service clients try to locate

them. This imposes an always-on requirement for service providers, which is unfeasible for battery-powered devices (Requirement R2).

3. *Context-aware queries*. In the current standard, the only available selection criterion is the service type, while additional descriptions of services can be added as part of TXT records. Therefore, in order to select a service with a specific context, as location, a service client must first gather the descriptions of all services of the same type, and then select the most appropriate one. In large networks, with many service instances with the same type, this generates a lot of traffic (Requirement R1).

4. *Packet size*. Due to the expressive nature of the protocol, RRs are relatively large, and require packet fragmentation from lower layers. Such fragmentation is generally unwanted in LLNs, as end-to-end delivery of fragmented packets is difficult. As shown in the previous section, fitting all RRs for a single service description would significantly reduce the overhead traffic associated with service discovery (Requirement R2).

Table 4.3: Code and memory footprint of different mDNS/DNS-SD implementations, in bytes.

Implementation	Code (ROM)	Memory (RAM)
Bonjour by Apple	500KiB [1]	/
Ethernet Bonjour for Arduino	14KiB	/
uBonjour for Contiki [KK12b]	7.69KiB	0.4KiB
mDNS/DNS-SD for Contiki [2]	6.51KiB	0.7KiB

[1] Based on the size of mDNSResponser.exe on 64 bit platforms. Memory information is unavailable.
[2] Available at https://github.com/mstolikj/contiki

We have solved the first problem by implementing a subset of the protocol for resource constrained devices (Table 4.3). The implementation contains only the basic features of the protocol - an engine and an API for publishing and discovering services, without additional features as multi-interface support and caching. The implementation contains the minimum amount of features required for participation in the mDNS/DNS-SD protocol, and is interoperable with other existing implementations. The implementation is open source, and feature-wise, it is comparable to the proprietary implementation described in [KK12b]. For the second problem, we propose a proxy architecture, explored in Section 4.4. For the third problem, we develop an extension of the protocol, which al-

lows queries to include context, as described in Section 4.5. Finally, we present ideas for solving the last problem in Section 4.6.

4.4 Proxy support for sleeping nodes

One of the drawbacks of practical implementations of mDNS/DNS-SD is that service providers are assumed to be constantly online. This comes from the distributed nature of the protocol: if a service provider is not online when a query for one of its services arrives, it will not be able to respond to it, thus its services will be undetectable. Additionally, in order to be able to quickly adapt to network changes, such as service providers leaving from the network, services are advertised with relatively short time-to-live intervals (2 minutes on LANs). Frequent messaging is an unwanted feature for LLNs due to the limited battery life of the participating devices. As illustrated in [TL09], this behaviour introduces a trade-off between signalling frequency, i.e. increased traffic in the network, and the risk of discovering non-existing services.

LLNs often include devices low radio duty cycles, or so called Sleeping Service Providers (SSPs). SSPs regularly turn off their radios, in order to reduce energy consumption. Therefore, during these offline periods their services are undetectable via the current mDNS/DNS-SD protocol. One approach to facilitate SSPs is to introduce proxy servers on high capacity (non battery powered) devices. Then, proxy servers can take over discovery functionality from SSPs. With proxy servers, we create an overlay network, with service discovery taking place between service clients and high capacity devices, and SSPs only limited to registering services with a proxy servers. This enables a more flexible deployment approach, and the possibility to create a mixed duty cycled/non duty cycled network.

Of course, proxy servers can be used for more tasks besides service discovery. For instance, CoAP relies on proxy servers to provide translation of CoAP messages to HTTP messages, along with caching support. However, due to the separation of layers, in this chapter we only consider proxy support for service discovery, and leave the option to extend proxy servers with additional capabilities as future work.

Within mDNS/DNS-SD, a key factor to include support for proxy servers is to develop a protocol for delegation of service descriptions from SSPs to proxy servers. We consider two implementations of such a protocol:

active and passive proxy delegation protocol. The distinction between the two protocols is based on the role of the SSP in the proxy selection phase. Both approaches are explained in the forthcoming sections.

4.4.1 Active proxy delegation protocol

In the active proxy delegation protocol, the SSP is the driving party during the delegation process. Proxy functionality is another service in the network, with multiple proxy servers hosting different proxy service instances. The SSP initiates the protocol by first searching for a proxy server. After the SSP has selected a suitable proxy server instance, it registers with it. Depending on the messaging used, the registration can be preceded by route discovery. Then, upon receiving a response from the proxy server, the SSP either begins its sleep cycle or tries to register with a different proxy server. The registration protocol itself is outside of the mDNS/DNS-SD specification and varies between implementations. In this work, we use the Bonjour implementation by Apple (Figure 4.7).

The Bonjour active proxy delegation protocol intends proxy servers to be located on fixed infrastructure devices as wireless routers, TV boxes or servers. Proxy servers advertise their service using the *_sleep-proxy._udp* service type. SSPs register with a proxy server using the Dynamic DNS Update (DDNS) protocol [RFC6891]. The registration message is a unicast DDNS packet, which contains all RRs which should be hosted at the proxy server, and an EDNS0 RR which specifies the lease time of the delegation [CK06], and ownership information of the SSP [CK09]. The ownership information is used to transfer the MAC addresses of the SSP to the proxy server. In the Bonjour implementation, this address is used to both intercept messages destined for the SSP while it is not available, as well as for waking up SSPs using the Wireless Multimedia Extension of 802.11e for wireless SSPs, or using wake-on-lan packets for wired SSPs. Since we focus on LLNs, using different protocol stacks, we have not implemented these features. The proxy server server always returns a unicast response, which informs the SSP whether the delegation request was accepted.

4.4.2 Passive proxy delegation protocol

The active proxy delegation protocol requires several messages to be exchanged before the delegation takes place. In order to reduce traffic,

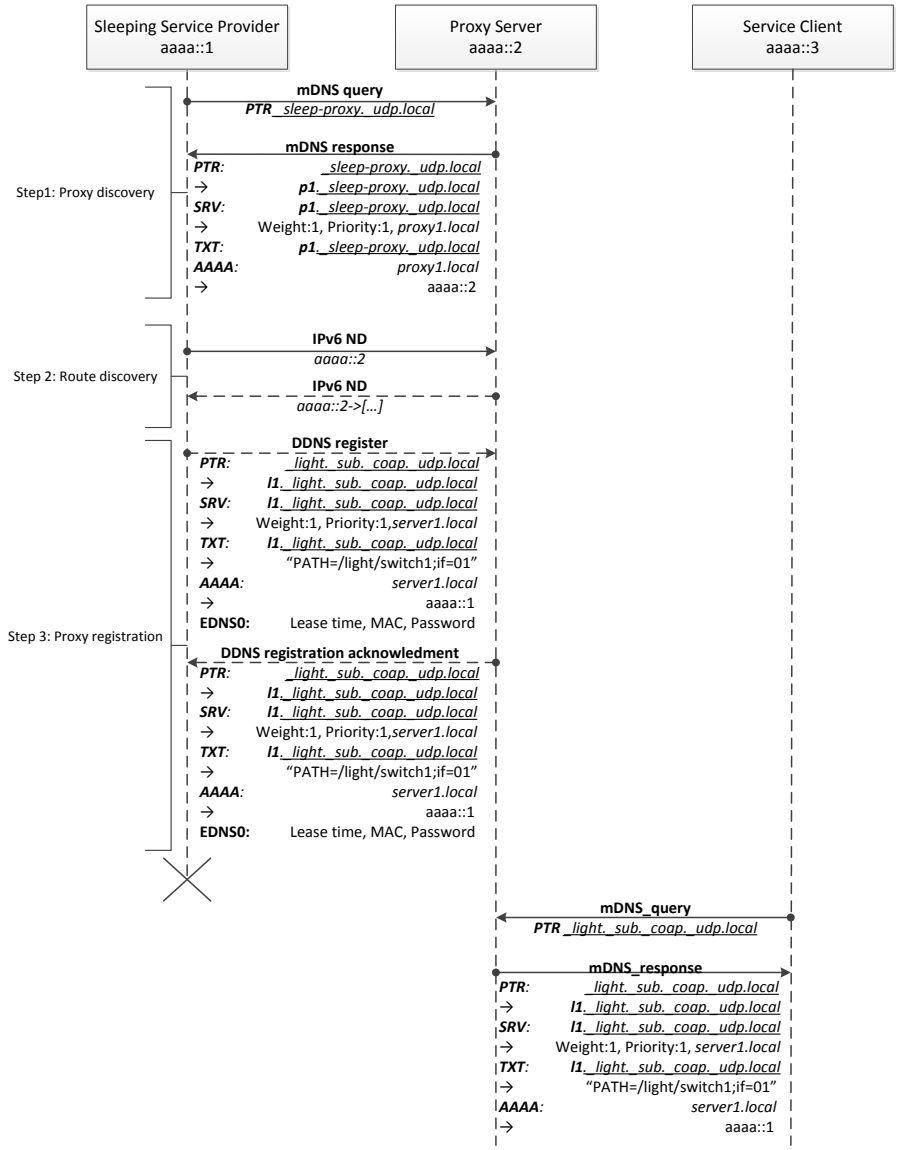


Figure 4.7: Active proxy delegation protocol in mDNS/DNS-SD. The service provider selects a proxy service and registers with it. Afterwards, the proxy server responds on behalf of the service provider. Note: full lines portray multicast messages; dashed lines portray unicast messages.

we propose a new protocol, called the passive proxy delegation protocol, where the registration request is embedded in the service advertisement. As shown in Figure 4.8, the SSP adds a parameter in the TXT RR of

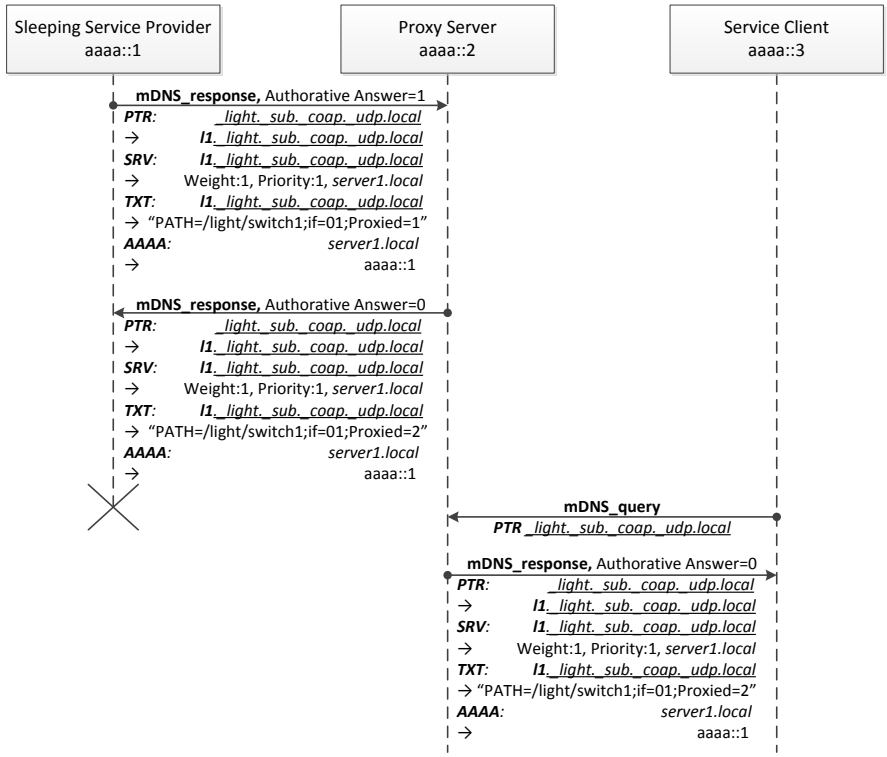


Figure 4.8: Passive proxy delegation protocol in mDNS/DNS-SD. The service provider indicates that it wants to be served by a proxy server in the service advertisement. This request is processed by the proxy server, and from there on, it starts responding on behalf of the service provider. The request is acknowledged by re-sending the advertisement. All messages are multicast.

its service description. This parameter is a signal for proxy servers to interpret the service advertisement as a registration request. The proxy server that decides to serve the request acknowledges the delegation by re-sending the service advertisement. The distinction between advertisements originating from the SSP and cached advertisements from the proxy server is done using the Authoritative Answer (AA) bit. The purpose of the re-transmission is two fold: 1) the SSP knows that someone has handled its request and can start its sleep cycle; and 2) other proxy servers know that they need not process that advertisement. The protocol can be further optimized by re-transmitting only the first SRV RR in order avoid unnecessary distribution of large (fragmented) messages. The SRV RR is unique to the SSP and can be undoubtedly interpreted by the SSP and by other proxy servers.

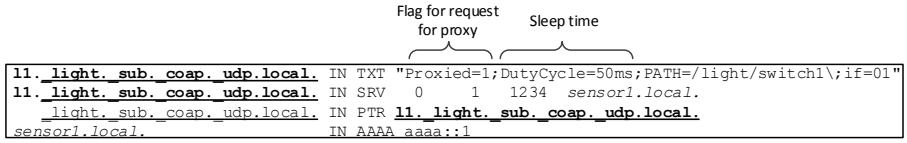


Figure 4.9: Embedded registration request for the passive proxy delegation protocol.

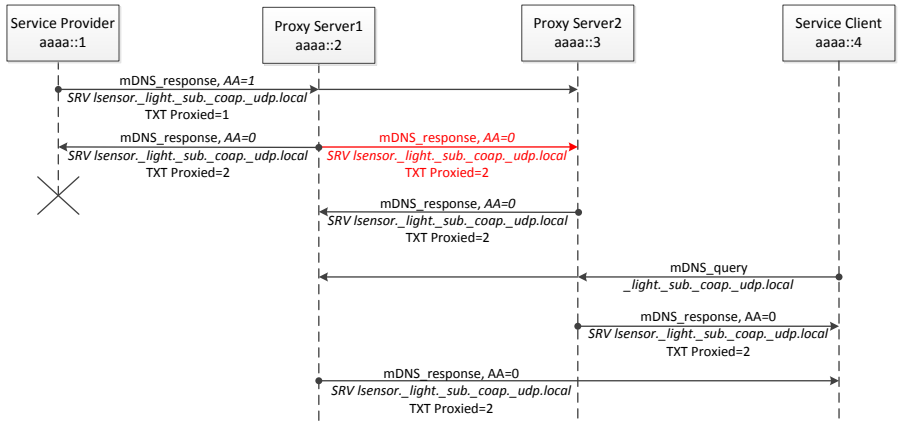


Figure 4.10: Possible double proxy registration using the passive proxy delegation protocol. If the re-sent advertisement by the first proxy server does not reach the other proxy server (marked red), another proxy will register the same service. As a result, queries for the service will be responded to twice.

For additional functionality, the proxy server needs to know the duration of the sleep cycle of the SSP. With the active proxy delegation protocol, this information is sent directly to the proxy server within the registration message, in the form of the lease time. With the passive proxy delegation protocol, it has to be added within the service advertisement. This information, along with the registration request, can be added as additional parameters within the TXT description of the service advertisement (Figure 4.9). Similarly, other required parameters, such as the MAC address, can be transferred. The compact nature of these two descriptions is of paramount importance since large service advertisements can lead to packet fragmentation.

4.4.3 Reliability

The active proxy delegation protocol does not significantly suffer if advertisement messages are lost. Simply, the proxy registration phase

has to be repeated, delaying the start of sleeping time of the SSP. On the other hand, message loss in the passive proxy delegation protocol can lead to double registrations, as shown in Figure 4.10. While this state is not functionally wrong, it leads to distribution of unnecessary messages in the network. A possible solution to this problem is to use the start up conflict resolution protocol defined in the mDNS specification, which would resolve the proxy assignment between two or more competing proxy servers.

4.4.4 Evaluation

We compare the performance of the active and passive proxy delegation protocols analytically and in simulation. We are interested in the memory footprint, the delay and the energy consumption for a service provider to register a service with a proxy server. For efficient low-power operation, we assume that ContikiMAC [Dun11b] is used as a Radio Duty Cycling (RDC) protocol.

4.4.4.1 Memory footprint

We implement the active and passive proxy delegation protocol in the Contiki operating system [DGV04], on top of our own mDNS/DNS-SD implementation. The size of the modules, compiled using the msp430-gcc (GCC) 4.5.3 compiler for the Crossbow TelosB nodes, are shown in Table 4.4.

Both the client and server components of the passive proxy delegation protocol are smaller than the corresponding components for the active proxy delegation protocol. This difference is due to the additional complexity required for implementing the DDNS protocol for the proxy registration. Even though the packet format is similar in DDNS and mDNS, additional resources are used for establishing the connection for the DDNS update protocol, which increase the footprint of the active proxy delegation protocol.

4.4.4.2 Analytical evaluation

The active proxy delegation protocol requires 3 multicast frames for discovering the proxy server. After the description of the proxy server has

Table 4.4: Code and memory footprint of different components for proxy registration, in addition to the mDNS/DNS-SD implementation (bytes)

Component	Proxy protocol	Code	Memory
Service provider	active	1.484	66
Proxy server	active	1.268	452
Service provider	passive	1.136	90
Proxy server	passive	902	432

been found, a route between the SSP and the proxy server is required. In a single hop IPv6 network, the route discovery is realized through the Neighbor Discovery Protocol [RFC4861] by sending a neighbor solicitation, which fits in one frame. If the neighbor has been previously known and its reachability has to be verified, a unicast frame is sent. Otherwise, another multicast frame is used. The response of the solicitation comes back in the form of a unicast neighbor advertisement. Finally, the registration protocol requires 3 unicast frames for sending the registration and 3 unicast frames for the response.

In a single hop network, a multicast transmission is essentially a broadcast. ContikiMAC implements broadcast traffic by repeating the message over the entire wake-up interval. On the other hand, unicast traffic is implemented by repeating the message until an acknowledgment is received, or an entire wake-up interval completes. Therefore, the time for sending one broadcast frame (t_b) is equal to the duration of the wake-up interval (w), plus any additional back-off by the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) protocol (t_o). The back-off is a factor of w. The time for sending an unicast frame (t_u) equals the duration of the CSMA/CA back-off plus the transmission time. In the best case, as when bursts of frames are sent, the sender and the receiver are in sync. Then, the frame would be transmitted once, and the receiver would immediately acknowledge it (t_{min}). In the worst case, the transmission time is equal to the wake-up interval (w).

Therefore, the delay for completing the active proxy registration protocol can be estimated as: $D_{ap} = t_{sd} + t_{nd} + t_{rg} \approx 3(t_b + t_o) + [t_o + t_b + t_o + t_u] + 2(t_o + t_u + 2t_{min})$, where t_{sd} is the time to discover the proxy server, t_{nd} is the time to discover a route and t_{rg} is the time to perform the registration.

The passive proxy delegation protocol uses only multicast frames. Both the initial service advertisement and the repeated service advertisement are sent via 2 multicast frames. The delay can then be estimated as: $D_{pp} \approx 4(t_o + t_b)$.

We can simplify the model by assuming that there is no message loss and there is no interfering traffic. In this way we prevent any retransmissions by the CSMA/CA protocol and we limit the duration of the back-off to at most w. Both of these factors influence the delay, as verified by the simulations.

The lower bound of both proxy protocols with the simplified model can then be found by assuming that no back-off takes place (i.e. $t_o \approx 0$) and that the sender and receiver are in perfect sync ($t_u \approx t_{min}$). Similarly, the worst case would then be if the back-off is maximum ($t_o = w$) and that the sender and receiver are off sync by w during the first unicast transmission. Therefore, the bounds of the delay of the active and passive proxy delegation protocol are: $D_{ap} \in [4w + 7t_{min}, 14w + 4t_{min}]; D_{pp} \in [3w + t_{min}, 8w]$. The calculated bounds are shown in Table 4.5.

Table 4.5: Upper and lower bound on the proxy registration protocols and the measured simulated results for an IEEE 802.15.4 channel with no loss and no background traffic.

Wake-up frequency	w	Proxy type	Lower bound	Upper Bound	Simulation average
4Hz	250ms	active	1.03s	3.52s	2.13s
		passive	0.75s	2.00s	0.98s
8Hz	125ms	active	0.53s	1.77s	1.20s
		passive	0.38s	1.00s	0.52s
16Hz	62.5ms	active	0.28s	0.89s	0.67s
		passive	0.19s	0.50s	0.29s

As expected, due to the smaller number of messages exchanged, the passive proxy delegation protocol finishes much faster than the active proxy delegation protocol. The different wake-up intervals only increase the gap between the two protocols. Furthermore, due to the predictable behaviour of the broadcasting protocol in a single hop network, it is easier to estimate the delay of the passive proxy delegation protocol.

The model becomes more complex in a multi-hop network, due to the uneven load between multicast and unicast messages. The performance then depends on the implementation of the underlying multicast protocol

and the role of packet fragmentation, which is outside of the scope of this chapter.

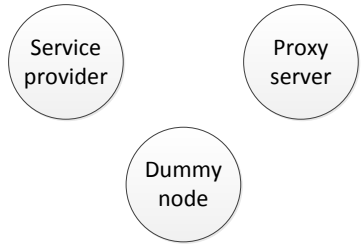


Figure 4.11: Test scenario for service discovery. All nodes are within the same broadcast domain.

4.4.4.3 Simulation

We simulate a single hop scenario in Cooja [Ost+06], a cross-level simulator for the Contiki operating system. Cooja internally uses the MSPsim device emulator for cycle accurate Crossbow TelosB emulation, as well as a symbol accurate emulation of the CC2420 radio chip. The test network consists of three Crossbow TelosB nodes - an SSP, a proxy server, and a dummy node, all located in the same single-hop 6LoWPAN network. The SSP advertises one service, described using four RRs: PTR, SRV, TXT and AAAA. All four RRs are stored in a single DNS packet, which is then fragmented into two 802.15.4 frames for transport. All nodes use the CSMA/CA Medium Access Control (MAC) protocol together with the ContikiMAC [Dun11b] RDC protocol with wake-up frequencies of 4, 8 and 16 Hz, which results in wake-up intervals of 250ms, 125ms and 62.5ms, accordingly.

For radio propagation, we use the Unit Disk Graph Radio Medium (UDGM) model for radio propagation, with constant loss probability. In separate simulations, we vary the packet delivery ratio between 80 and 100%, at 2% increments. All charts show the mean values of 1.000 runs, and the error bars correspond to the 95% confidence interval of the mean.

The simulation starts with all nodes being online. At second 5, the SSP advertises its service. At second 7, the SSP starts delegating the service description to the proxy server using one of the previously described protocols. After the delegation finishes, the SSP turns off its radio, and the simulation is stopped. The dummy node does not participate in the

proxy delegation protocol, but it overhears all traffic in the network. We use it to show the impact of the protocols to nodes in the vicinity.

Due to packet loss, the proxy delegation may not succeed in its first iteration. Therefore, we implement repetitions of individual stages of the delegation protocols based on the expiry of fixed timeouts. The timeout for completion of step 1 from the active proxy delegation protocol and the entire passive proxy delegation protocol is set at 5 times the RDC wake-up interval. The repetition of step 2 of the active proxy delegation protocol is dictated by the neighbour discovery protocol, and the timeout is fixed at 10 seconds. Finally, the timeout of the entire step 3 is set at 2 seconds. This should be enough to capture any retransmissions of unicast frames by the CSMA/CA protocol.

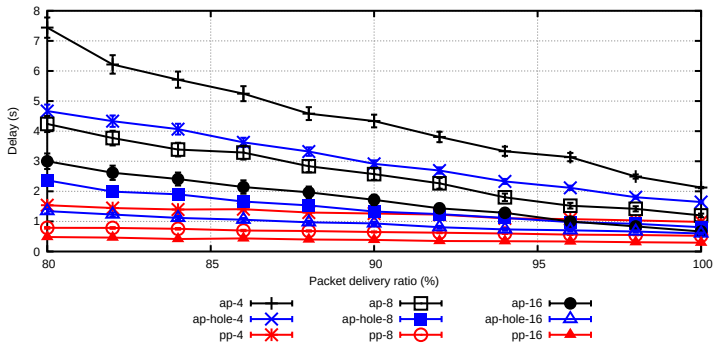


Figure 4.12: Required time (delay) for completing the active and passive proxy delegation protocol, as measured by the service provider. *ap* and *pp* refer to the active/passive proxy delegation protocol, with 4, 8 and 16Hz wake-up intervals for the ContikiMAC RDC protocol. *ap-hole* refers to the optimized version of the active proxy delegation protocol.

We compare the performance of the active (**ap**) and passive proxy (**pp**) delegation protocol. To verify the impact of the neighbour discovery protocol (Step 2 of the active proxy delegation protocol from Figure 4.7), we also implement an optimized version of the active proxy delegation protocol (**ap-hole**), where the neighbour discovery stage is skipped. In this optimised version, the link layer address of the proxy server is generated from the last 8 bytes of the IPv6 address, present in the AAAA RR of the *_sleep-proxy* service description.

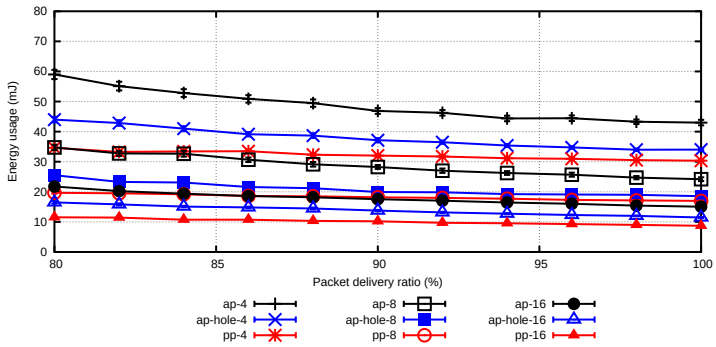
Figure 4.12 shows the delay, i.e. the time elapsed from starting the proxy delegation protocol until its completion. As expected, increasing the wake-up interval results in much higher delays. With all three different

wake-up parameters, due to the smaller number of messages, the passive proxy delegation protocol finishes much faster than the active proxy delegation protocol. The improvements vary depending on the RDC wake-up interval and the packet loss, from 2 to 6 fold. The optimized active proxy delegation protocol also converges faster, up to 2 fold. The difference between the two versions grows as the packet delivery ratio drops. Still, even the optimized version requires more time to complete than the passive proxy delegation protocol.

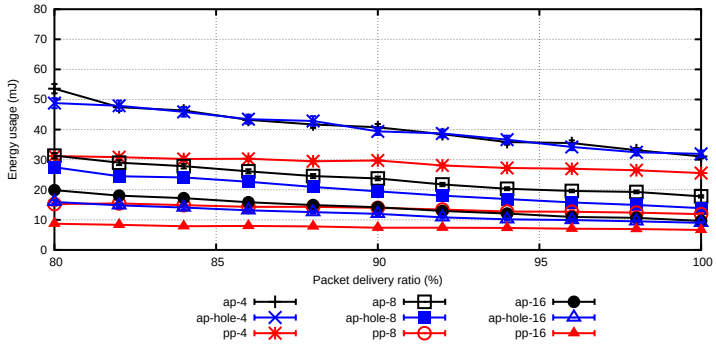
Figure 4.13 shows the energy usage of the SSP and the proxy server during the proxy delegation. These measurements are profiled using the software power profiler Powertrace [Dun11a]. Contrarily to expectations, increasing the wake-up interval actually increases the energy usage for the proxy server and the SSP. This behaviour is due to the RDC protocol: ContikiMAC makes senders do more work than receivers during transmissions.

The energy usage of the passive proxy delegation protocol is lower compared to the active proxy delegation protocol, though the differences are not as high as with the delay. We attribute this behaviour to the efficiency of the ContikiMAC protocol. Furthermore, in the active proxy delegation protocol, if a message loss occurs during the neighbour discovery phase, the sender will be silent for the entire timeout period, which introduces a large delay, but not much energy usage. This is visible in the energy usage of the active proxy delegation protocol and the optimized version with a 4Hz wake-up interval. Even though the difference between the two in terms of delay is large, they consume similar amounts of energy at the proxy server side.

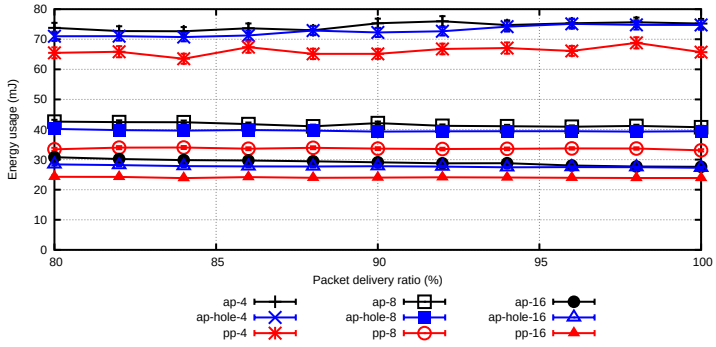
Finally, Figure 4.13c shows the average energy usage of the dummy node in the network. The energy usage is measured during the entire simulation, and includes the mDNS initialization stage, where every node advertises its host name and address. The passive proxy delegation protocol requires less energy due to the smaller number of messages during both the initialization phase and registration phase. The passive proxy is silent during the initialization phase, while the active proxy advertises the *_sleep-proxy* service. Surprisingly, the neighbour discovery does not significantly impact the active proxy delegation protocol in this aspect. Namely, the energy footprint of the active proxy delegation protocol is close to the optimized version.



(a) Sleeping Service Provider



(b) Proxy Server



(c) Overhearing Node

Figure 4.13: Energy usage for completing the active and passive proxy delegation protocol. *ap* and *pp* refer to the active/passive proxy delegation protocol, with 4, 8 and 16Hz wake-up intervals for the ContikiMAC RDC protocol. *ap-hole* refers to the optimized version of the active proxy delegation protocol. The energy usage of the overhearing node (c) is measured for the entire simulation duration.

4.4.4.4 Summary

From the evaluation, we can conclude that in a single-hop network, the passive proxy delegation protocol has better performance when compared to the active proxy delegation protocol. This is evident in both memory usage, delay and energy consumption. For a multi-hop network, the choice between the two protocols depends on the network topology, the implementation of the multicast forwarding algorithm, and whether a route between the SSP and proxy server is known beforehand. We leave this as future work.

4.5 Support for context queries

The DNS-SD protocol was designed for discovering all services instances of a given type within a certain logical domain. The protocol assumes that within the given logical domain, either all service instances of the same type provide the same functionality and selecting any of them is enough, or an end-user can select one. In the former case, when a single service needs to be selected, the priority/weight fields of the SRV records play an important role. In the latter case, the end-user can select a preferred service instance based on the service instance name or based on the human-readable description, added as payload in the TXT RR. However, none of these methods can be added within the queries sent out by service clients, i.e. they cannot be used to reduce the number of responses. Moreover, the interpretation of the TXT RR is outside of the DNS-SD specification.

In the IoT, services of the same type will be available in abundance. As a result, queries for common service types will result in many responses, which poses a serious burden to the network. Moreover, in M2M communication, as between sensors and actuators, the selection of a service has to be done automatically, using information present in service descriptions. In such scenarios, the logical domain criterion is not enough to discriminate between services of the same type. Therefore, the DNS-SD protocol has to be adapted to provide means for stricter selection of services based on criteria present in queries.

Currently, two approaches have been proposed. The first approach, described in [SL12], customizes DNS-SD functionality for building control, where services are selected based on location, general type and

subtype. The location information is added in each service description as part of the domain name, while the protocol is added via the type/subtype options. For instance, a light switch located in floor 1, office 1 in the TU/e MetaForum building would have a PTR RR with the name *_OnOff_light._sub._bc._udp.o01.f1.mf.tue.nl*. In the remainder of the text, we refer to this method as the building-control approach.

The second approach, described in [Agg14], uses TXT RRs inside queries as a way to impose constraints for service selection. The TXT record contains key-value pairs of entries which specify features required by the services. Then, every SP responds only if the TXT RRs associated with its service, contain all entries in the query TXT RR. For example, for the same service as the previous approach, the service client would send a query with two RRs: one PTR, with *_light._sub._bc._udp.tue.nl* as name, and one TXT, with entries: *type="OnOff", building="mf", floor="1", office="01"*. Of course, all location entries can be collapsed into one, as in the building-control approach.

Both approaches are an improvement over the current DNS-SD standard. However, neither is ideal. While the former approach lacks flexibility and is tightly coupled with location as the primary discriminator, the latter approach imposes additional overhead in the size of the queries. Therefore, we propose a mixture of the two approaches: a service description model based on context tags, added as names in PTR RRs.

4.5.1 Context tag descriptors

We define that any given service instance can be associated with a set of context tags. A context tag is an atomic descriptor of one context property. We assume an inclusive model: the association of a context tag with a service instance represents that the service instance has that specific context, while the lack of a context tag represents that the associated service instance does not have that specific context.

Service discovery then consists of sending one or more queries, listing a combination of wanted/unwanted context tags. As a response, a set of services is expected, whose set of tags satisfy the queries. We use boolean logic to express the queries, using conjunction ($\wedge$), disjunction ($\vee$) and negation ($\neg$) operators on context tags. The discovery model then consists of a:

- Set of service instances S,

- Set of tags T,
- Mapping function $\pi : S \to \mathcal{P}(T)$, that describes which context tags are associated with a given service,
- Set of queries, defined by the grammar $Q := t|Q \wedge Q|Q \vee Q|\neg Q$, with atoms $t \in T$,
- Service discovery function $\sigma : Q \to \mathcal{P}(S)$, the mapping between queries and a set of matching service instances.

The discovery function is then defined as:

$$\begin{gathered}\sigma(t) = \{s \in S | t \in \pi(s)\} \\ \sigma(p \wedge q) = \sigma(p) \cap \sigma(q) \\ \sigma(p \vee q) = \sigma(p) \cup \sigma(q) \\ \sigma(\neg t) = \{s \in S | t \notin \pi(s)\} \\ \sigma(\neg(p \wedge q)) = \sigma(\neg p) \cup \sigma(\neg q) \\ \sigma(\neg(p \vee q)) = \sigma(\neg p) \cap \sigma(\neg q)\end{gathered}$$

In [Buc14], we explore several options how the context model can be implemented as service selection criteria using DNS-SD. There, the emphasis is on the trade-off imposed by the added expressive value (i.e. more specific queries) and the increased complexity. For brevity, here we only describe the most promising one: complete predicates as selection criteria.

4.5.1.1 Predicates as selection criteria

With this approach, we encode the entire predicate (σ) inside the PTR name, and use it as criteria for service selection. The predicate is parsed and evaluated at each service instance individually. If the given service instance satisfies the predicate, a response is sent back to the client.

To preserve compatibility with DNS-SD, all context tags and operators have to be in human-readable form (i.e. ASCII or UTF characters). Therefore, we use the asterisk (*), dot (.) and hyphen symbol (-) for disjunction, conjunction and negation operator, respectively. For simplification purposes, we store the predicate in Disjunctive Normal Form (DNF). This enables us to break long formulas in separate DNS RRs. Finally, after the predicate, we add the service type and the logical domain, again to preserve backward compatibility.

An obvious weakness of this approach is that it requires more complex behaviour on the service provider side. Namely, the SP has to be able to parse and evaluate a boolean predicate. Therefore, we advise against using nested terms, and utilizing only simple predicates.

An additional problem is posed by lengthy predicates. The DNS-SD specification limits domain names to 253 characters. Therefore, long predicates with many atoms would have to be broken in multiple RRs, which results in additional communication overhead and processing.

4.5.1.2 Comparison

To compare the three approaches, we use the following scenario. Assume a network of one service client and two service providers, with features as shown in Table 4.6. The service client wants to discover a light switch for blue lights in office 1.

Table 4.6: Description of test scenario.

Property	Light switch 1	Light switch 2
Service type	_light._sub._coap._udp.<domain>	
Location	MetaForum, Floor 3, Office 1	
Color	Blue	Red
Resource path	/light/switch1	/light/switch2

For the building-control approach, one additional PTR RR needs to be created, with the location included in the domain name. The color description is then part of the TXT RR. The service client first needs to discover all light switches for office 1, and then from the responses, check the TXT RRs to find the service instance for blue lights. The complete message exchange is shown in Figure 4.14.

When TXT RRs are used as part of the query, all context features are encoded inside the TXT RR. The service client sends the TXT RR with the wanted features together with the PTR RR as a query. Only one response is expected, as the query is very specific. The complete message exchange is shown in Figure 4.15.

Finally, with our approach, only one PTR RR is sent as a query, containing all requested contexts. As previously, one response is expected. The complete message exchange is shown in Figure 4.16.

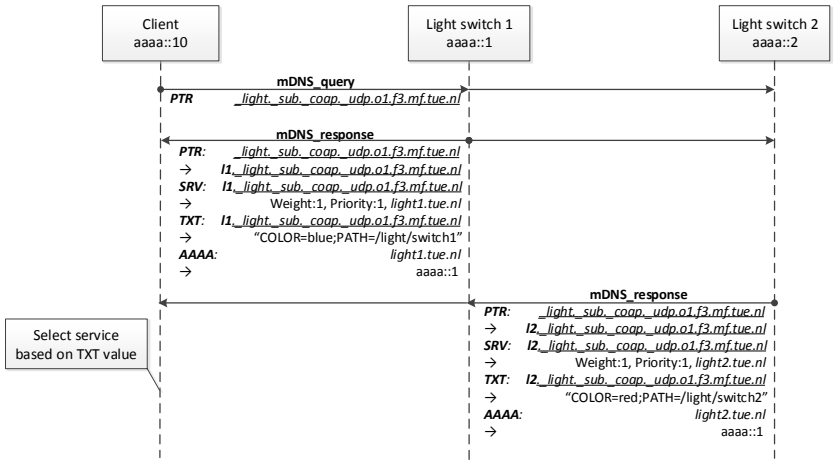


Figure 4.14: DNS-SD message exchange during discovery of a particular light service, using the building control approach.

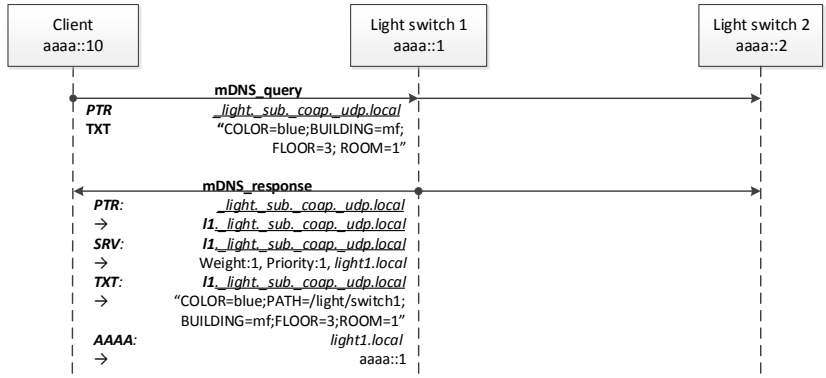


Figure 4.15: DNS-SD message exchange during discovery of a particular light service, using TXT records as part of queries.

This example shows the design trade-offs in the three approaches. On the one hand, the building-control approach is simple to implement. However, it does not scale well and it lacks flexibility. As shown, discriminating responses based on a property other than location and service type has to be done by the service client. Furthermore, the granularity of the location has to be known in advanced, as separate PTR RRs need

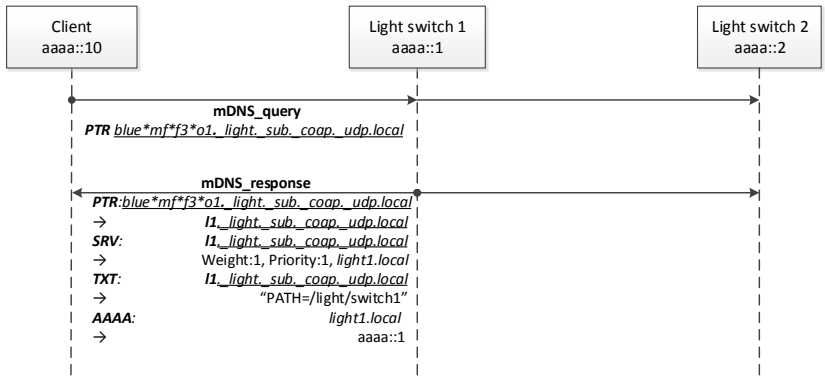


Figure 4.16: DNS-SD message exchange during discovery of a particular light service, using predicates inside PTR RRs as part of queries.

to be created for any particular location contexts (e.g. finding all light switches in a building/floor/office require three different PTR RRs).

On the other hand, the latter two approaches enable more flexible filtering of services during the query phase, at the cost of additional complexity at the service provider side. In the TXT query approach, the service provider has to process both PTR host names for matching, as well as all the entries in the TXT RRs, and match them with any local services. Similarly, in the predicate PTR approach, the service provider would have to both parse the predicate, and evaluate it for every local service. However, we believe than the benefits of using such approaches, in early reduction in the number of expected responses, outweigh the implementation issues. A relieving circumstance is that in a fully distributed environment, the number of local services per provider would not be large, so the processing required is not significant.

4.6 Future work: Packet size

Table 4.7 shows the minimum size of the DNS-SD RRs associated with a single service description. It is obvious that even in the best case, with human unreadable single-letter service/host names, all four RRs are larger than the maximum available payload size in IEEE 802.15.4 with 6LoWPAN, and require fragmentation.

Table 4.7: Size of DNS Resource Records

RR Type	PTR	SRV	TXT	A/AAAA
Name	query name [1](1) >14B	service name 2B (ptr→(2))	service name 2B (ptr→(2))	host name 2B (ptr→(3))
Type	2B			
Class	2B			
TTL	4B			
Length	2B			
Priority	-	2B	-	-
Weight	-	2B	-	-
Port	-	2B	-	-
Data field	service name [2](2) >5B [4]	host name [3](3) >5B [5]	Arbitrary field 0-1300B	IPv4/6 Address 4/16B
Total	>29B	>23B	>12B	16/28B

[1] <query name>.<tcp/udp>.local
[2] <service name>.<tcp/udp>.local
[3] <host name>.local
[4] <service name>+ ptr→(1)
[5] <hname>+ ptr→(1)

In [KK13], a compression scheme called Adjustable DNS Message Compression (ADMC) Enhanced is proposed, which aims to reduce the combined size of DNS-SD packets using four techniques (Figure 4.17). In the first step, RRs are combined to use the existing DNS name compression method. Then, the CLASS and TTL fields of all four RRs are collapsed into one. This is a reasonable assumption, since they usually hold the same value for all RRs associated with the same service. Next, the address in the AAAA record is compressed, by linking it to the address assumed to be present in the 6LoWPAN packet. Finally, 6LoWPAN compression (IPHC/NHC) is applied, yielding a DNS packet which is potentially less than 60 bytes and fits in one 6LoWPAN frame. However, this approach relies on having a relatively small service description, which is contradictory to the context-aware query support described in the previous section.

Therefore, we believe that additional savings can be reached if service/host names are encoded in binary format inside of RRs. While DNS-SD advocates the use of human-readable names, this is not a necessity in M2M communication, as in the IoT. Furthermore, the encoding/decoding will be hidden by the DNS-SD API, which is responsible for translating RRs to their full, human-readable form.

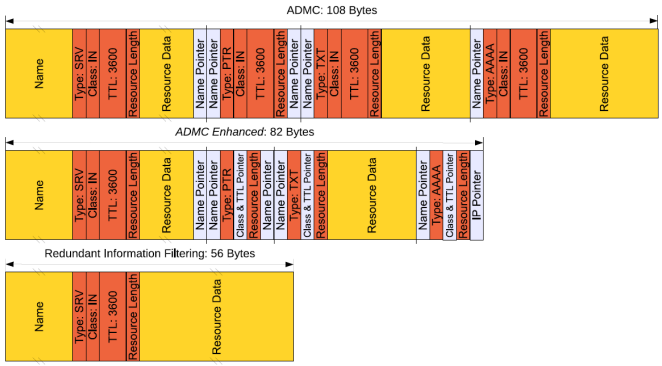


Figure 4.17: DNS compression as described in [KK13]. The topmost figure (ADMC) shows packet size when only DNS name compression is used. The middle figure (ADMC enhanced) shows packet size when the class and TTL fields are encoded as pointers, and the IP address is reconstructed from the network layer header. The bottommost figure (Redundant information filtering) collapses all repetitive fields as one, and is incompatible with the current DNS-SD standard. Figure reprinted from Ronny Klauck and Michael Kirsche. "Enhanced DNS Message Compression - Optimizing mDNS/DNS-SD for the Use in 6LoWPANs". In: *Workshop on Sensor Networks and Systems for Pervasive Computing*. PerSeNS. 2013.

A similar approach was recently taken for reducing the size of payloads for a network management interface for constrained devices using CoAP [Sto+15]. The authors use Concise Binary Object Representation (CBOR) [RFC7049] for encoding JavaScript Object Notation (JSON) [RFC7159] objects as part of the CoAP payload. In addition, commonly used strings, as paths as part of URIs, are first hashed to generate a 32bit identifier, and registered at all parties. Then every reference to a known string is replaced with the hash. A similar approach is viable for DNS-SD, where the service types, context tags, host names and their combination, can be exchanged in hashed form. In the best case, all names can be replaced with 4 byte hashes or 2 byte DNS compression lookups, which is enough to avoid packet fragmentation.

4.7 Conclusion

In this chapter we addressed research questions **RQ2** and **RQ4**, by analyzing the applicability of the mDNS/DNS-SD protocol for service discovery in the IoT. We identified several key problems in the existing

version of the protocol. Firstly, existing implementations are too large to implement on resource constrained devices. Secondly, the protocol requires service providers to be constantly online, which is unsuitable for battery-powered devices. Thirdly, the service selection capabilities of the protocol are not rigorous enough to discriminate between many service providers in large network. Lastly, the expressive nature of the protocol, with services described with long, human-readable domain names, results in large packets for discovery. In LLNs, due to the limited capacity of the media, large packets are usually fragmented into several smaller ones, and their delivery can be problematic.

To resolve the first problem, we developed a small implementation of the protocol, with a limited set of features. Then, for the second protocol, we proposed the introduction of proxy servers, which take over service discovery functionality from service providers. We presented two protocols for the delegation mechanism between the service provider and the proxy server. In the first, active proxy delegation protocol, the service provider actively searches and selects the proxy server for delegation. In the second, passive proxy delegation protocol, the service provider only signals its intention that it requires a proxy service: the proxy server picks up and processes the intention as a registration request. Both protocols were implemented in the Contiki operating system.

The simulations show that in a single hop network, the passive proxy delegation protocol converges faster, requires less energy and is smaller in code size when compared to the active proxy protocol. The improvements vary depending on the MAC protocol behaviour and the loss in the wireless medium. With a sender-initiated RDC protocol, the passive proxy delegation protocol can finish on average up to six times faster than the active proxy delegation protocol, consuming half of the energy.

Next, we proposed an extension of the protocol, which enables services to be described using additional context labels. With the extension, clients can then search for services which satisfy a predicated which contains a set of tags, connected with boolean operators. As a result, clients can discover very specific service instances among a group of many services of the same type.

As future work, we present ideas for reducing the size of packets associated with service discovery. This reduction would enable more efficient operation of the protocol in large LLNs.

5

TRICKLE-BASED PROTOCOLS

The system architecture and the application protocols described in the previous chapters assume that Low-Power and Lossy Networks (LLNs) have the capability to deliver unicast and multicast traffic. In the Internet of Things (IoT) domain, two standardized protocols are forerunners for those tasks: IPv6 Routing Protocol for LLNs (RPL) and Multicast Protocol for LLNs (MPL). Both protocols are based on the Trickle algorithm for efficient data dissemination in LLNs, which is the topic of this chapter.

Trickle aims to propagate information quickly in a network, while adapting transmission frequency based on the state of the network, and suppressing redundant transmissions. In this chapter, we analyse how the Trickle algorithm scales in size in different network topologies, and what is the impact of low power radios on its performance (Research questions RQ4 and RQ5). We verify our findings analytically and experimentally, and show how particular fallacies in the design of the Trickle algorithm, can affect either RPL or MPL.

The research presented in this chapter was done in collaboration with Thomas M. M. Meyfroyt. While most of the text was written as joint-effort, Thomas is the principal author of the analytical models presented in sections 5.4.2 and 5.5.2.

The chapter is based on the following publications:

- Milosh Stolikj, Thomas M.M. Meyfroyt, Pieter J.L. Cuijpers, and Johan J. Lukkien. "Improving the Performance of Trickle-Based Data Dissemination in Low-Power Networks". In: *European Conference on Wireless Sensor Networks*. EWSN. 2015, pp. 1–16
- Thomas M.M. Meyfroyt, Milosh Stolikj, and Johan J. Lukkien. "Adaptive Broadcast Supression for Trickle-Based Protocols". In: *IEEE Symposium on A World of Wireless, Mobile and Multimedia Networks*. WoWMoM. 2015

5.1 Introduction

Trickle [LPCS04] is a polite-gossip algorithm for propagating data in LLNs. Gossiping algorithms are used in computer networks to quickly propagate new information (gossip) by allowing each participant in the network to take part in spreading the gossip. In this context, politeness means that a participant will not spread a gossip if someone else already did. While Trickle was originally designed for propagating and maintaining code updates in Wireless Sensor Networks (WSNs), it has shown to be a powerful mechanism that can be applied to a wide range of protocol design problems, and therefore has been documented by the Internet Engineering Task Force (IETF) in its own Request For Comments (RFC) [RFC6206]. Furthermore, it has been adopted in two fundamental protocols for the IoT: RPL [RFC6550] and MPL [HK14]. These two protocols are of particular importance for the system architecture proposed in Chapter 2, since they enable basic type of communication patterns in LLNs. Therefore, one might consider Trickle as a new communication primitive for the IoT [Lev+08].

The Trickle algorithm is based on the premise that nodes independently decide if and when to transmit data. A node may decide to stay silent if it hears a few other nodes transmitting data which can deem its transmission redundant. However, if a node receives data which it deems to be inconsistent with its own local information, it will start communicating quickly to resolve the inconsistency. The concept of consistency is defined on a case by case basis, which allows Trickle to be implemented in many protocols. For instance, in RPL, Trickle is used to control the transmission of routing control data. There, a node can suppress its transmission if it hears a pre-defined number of routing control data from neighbouring nodes. An inconsistency is defined when a node changes its local routing information, and needs to quickly inform its neighbours of the change. Similarly, in MPL, Trickle is used to forward multicast packets in constrained networks. An MPL transmission of an already seen data packet is considered a consistency, while a new, unseen data packet is considered as an inconsistency.

While Trickle has been extensively studied, it still has weaknesses, which are explored in this chapter. We first analyse how Trickle scales in networks of varying density. We show that due to the static nature of its suppression mechanism, Trickle suffers from uneven load distribution. We demonstrate how this unfairness causes suboptimal RPL performance.

To resolve the issue, we propose an extension to Trickle, called *adaptive-k*, which adapts the suppression mechanism according to local density information (Requirement **R1**).

Second, we focus on the interplay between Trickle and the link layer. The link layer of low power radios inevitably introduces additional delays for transmission of data. Typical sources of delay are the Medium Access Control (MAC) protocols and Radio Duty Cycling (RDC) protocols. We show that due to the lack of feedback information between Trickle and the link layer, these interferences can cause inconsistencies within the Trickle algorithm. In specific topologies, this behaviour can cause starving, redundant messaging, or message loss. We use a data propagation method similar to MPL as a case study which demonstrates the negative effects. Finally, we propose an extension to the MAC protocol, called *Cleansing*, to resolve the interference (Requirement **R2**).

This chapter is organized as follows. We present the Trickle algorithm and its implementation details in RPL and MPL, in Section 5.2. Then, we give an overview of related work in Section 5.3. In Section 5.4 we take a closer look at the Trickle suppression mechanism and discuss why setting it correctly is difficult. Additionally, we propose the *adaptive-k* extension to the Trickle algorithm. We validate the extension using RPL as a case study. In Section 5.5, we give an overview of the protocols at the link layer which deliver Trickle messages. We present two scenarios how the behavior of these lower-layer protocols can disturb Trickle operation, and propose a solution at the link layer. We evaluate the effect of the proposed solution using MPL as a case study. Finally, we summarize our results in Section 5.6.

5.2 The Trickle Algorithm

Trickle has two main goals. Firstly, whenever new information becomes available in the network, it must be propagated quickly to all nodes. Secondly, when there is no update, communication overhead has to be kept to a minimum. The Trickle algorithm achieves this by moderating the number of packets that nodes generate with a "polite gossip" policy.

We now describe the Trickle algorithm as generalized in [MBBD14]. The Trickle algorithm is controlled via four configuration parameters:

- Threshold value k, called the redundancy constant.

- Minimum interval length $I_{\min}$.
- Maximum interval length $I_{\max}$.
- Fraction of the interval used only for listening η (default $\eta = 1/2$ [i])

Locally, each node in the network maintains a timer and three variables:

- Length of the current interval I.
- Counter c, for the number of received Trickle messages during the current interval.
- Transmission time t within the current interval.

The behavior of each node is described by the following set of rules:

1. Initially, $I = I_{\min}$, and a node starts a new interval.
2. At the start of a new interval, a node resets its timer, sets $c = 0$ and uniformly selects t to a value in $[\eta I, I)$.
3. When a node hears a message that it considers to be consistent with the information it has, it increments c by 1.
4. When a node's timer reaches time t, the node broadcasts its message if $c < k$.
5. When a node's timer reaches time I, it increases its interval length to $\min(2I, I_{\max})$ and starts a new interval.
6. When a node hears a message that it considers to be inconsistent with the information it has, then if $I > I_{\min}$ it sets $I = I_{\min}$ and starts a new interval, otherwise it does nothing.

Trickle only determines when nodes should transmit; the nature of the transmission (broadcast/unicast), the structure of the message, and the exact definition of what is a consistent transmission is given by the upper layers, i.e. the protocols where Trickle is used. For instance, in dissemination protocols, as multicast, transmissions are always broadcasts; a node receives a consistent transmission when a known data packet is

[i] The listen only interval was introduced as a parameter recently in [MBBD15]. The initial publication of the Trickle algorithm [LPCS04] only explores two options - having no listen only interval ($\eta = 0$) and having an interval of a half ($\eta = 1/2$). As the former option was proven to cause the number of transmissions by the algorithm to scale as $O(\sqrt{n})$ with network density ("short listen problem"), the initial version of the algorithm has a hardcoded listen only period of halft the interval. In the remainder of the text, unless specified, we assume $\eta = 1/2$.

received from another node, and an inconsistent transmission is received when a new, unseen data packet is received.

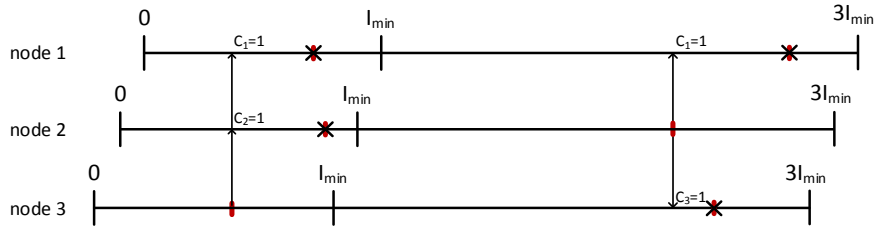


Figure 5.1: Example of three consistent nodes using the Trickle algorithm ($k = 1$, $I = I_{\min}$). The redundancy constant defines how many transmissions are expected per interval, while the minimum interval length defines the frequency of update/synchronization. In the first interval, node 3 is the first node to transmit. Since it sends consistent information, nodes 1 and 2 update their respective counters (c) to 1. As $k = 1$, they will suppress their transmissions. At the end of the interval, since no inconsistencies were detected, all nodes double their interval. Similarly, in the second interval, the transmission by node 2 suppresses nodes 1 and 3.

In Figure 5.1 an example is depicted of a network consisting of three nodes using the Trickle algorithm with $k = 1$ and $I = I_{\min}$ for all nodes. In practice, networks will generally not be synchronized, since synchronization requires additional communication and consequently imposes energy overhead. Furthermore, as nodes get updated and start new intervals, they automatically lose synchronicity.

To summarize, Trickle controls data dissemination on a per interval basis. The interval length provides a controllable parameter for the frequency of transmissions: lower interval lengths result in more frequent transmissions. During each interval, every node individually decides whether to transmit based on at least two criteria: the consistency state of the network (i.e. whether all nodes share the same information), and the number of received consistent messages from its neighbours (c), with respect to the redundancy constant (k). A new interval is started either immediately, if an inconsistency is detected and the current interval length is longer than $I_{\min}$, or at the end of each interval. If an inconsistency was detected, the length of the new interval is set to a pre-defined minimum value ($I_{\min}$). Otherwise, the rate of transmission is slowed down, by doubling the interval length, upto a pre-defined maximum value ($I_{\max}$).

The four Trickle parameters can be used to tweak the algorithm behavior according to specific scenarios, giving option for trading between redundancy, speed of propagation and risk of collisions. For instance, $I_{\min}$ provides a trade-off between speed of propagation and number of packets: lower values of $I_{\min}$ will make nodes transmit sooner, though with an increased risk of collisions, and therefore, additional transmissions. To prevent such scenarios, the Trickle RFC recommends setting $I_{\min}$ to a multiple of the worst-case link layer latency, defined as the time until the first link layer transmission of a frame, assuming an idle channel. Similarly, the redundancy constant k, provides a trade-off between communication overhead and robustness against message loss and networks with varying density. Typical values of the Trickle parameters for various protocols are given in Table 5.1.

Table 5.1: Default values of Trickle parameters in different protocols.

Protocol	k	$I_{\min}$	$I_{\max}$
MPL (control)	1	10 times worst-case link layer latency	300 s
MPL (data)	1	10 times expected link layer latency	$I_{\min}$
RPL (DIO)	10	8 ms	8.280 s
CTP [Gna+09]	$\infty(0)$	125 ms	500 s

5.2.1 RPL basics

RPL is a distance vector routing protocol for LLNs that uses the Trickle algorithm to build a Destination Oriented Directed Acyclic Graph (DODAG). The DODAG is a tree-like network-graph, rooted in a single node, in which all nodes learn a route towards the root node. While RPL supports both upwards and downwards routes, in this work we only focus on upward routes. The DODAG is shaped according to one or more Objective Functions (OFs), which can specify metrics for the cost of routes or give rules/constraints when building links.

The DODAG is built starting from the root as follows (Figure 5.2). The root advertises information about the graph using DODAG Information Object (DIO) messages, which are disseminated based on a Trickle timer. DIO messages contain information about the DODAG, its configuration parameters and the *rank* of the sender in the DODAG - a monotonically decreasing measurement indicating the distance from the sender to the root according to the objective function(s). Non-root nodes process these

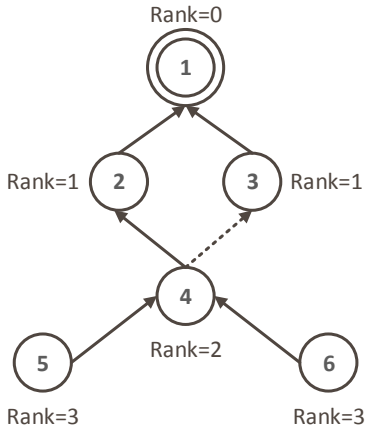


Figure 5.2: Example of a simple DODAG using hop count as an Objective Function. Node 4 has two parents, and selects node 2 as the preferred one.

DIOs and based on the objective function and/or some local criteria, decide whether to join the network. After joining, they establish which directly reachable nodes can forward data most efficiently towards the root (i.e. have the lowest rank). The best nodes form the *parent set*, while only one parent is actively selected for forwarding, named the *preferred parent*. Then, the nodes compute their own rank, and start transmitting DIO messages.

Once all nodes select a parent and become a part of the DODAG, we say that the DODAG has been formed. Whenever a node needs to send a message to the root, it sends it to its parent, which then forwards it to its own parent until it reaches the root.

The RPL specification defines the minimum set of consistency/inconsistency rules for the Trickle algorithm. A received DIO transmission is considered consistent if the rank of the sender does not cause a change in the local state of the receiver, i.e. the receiver keeps the same rank, parent set and preferred parent. An inconsistency must be considered when a node detects a problem in forwarding, suggesting a problem in the route; or a node joins a new DODAG version; or a node receives a multicast DODAG Information Solicitation (DIS) message, suggesting that a new node wants to join the DODAG. However, the RPL specification allows implementations to consider other rules as well.

5.2.2 MPL basics

MPL is a protocol for enabling multicast forwarding in LLNs. At the time of writing, it is still under development, so we present it as described in IETF draft version 9.

MPL is built around the premise that maintaining multicast routing topologies is unfeasible in LLNs. Therefore, it specifies how data should be disseminated to all designated MPL forwarders within a given MPL domain using the Trickle algorithm. MPL forwarders are nodes that are willing to route MPL messages. The MPL domain defines the scope of the multicast forwarding. Messages are generated by MPL seeders, which are MPL forwarders with a unique identifier. To distinguish between MPL messages and the initial messages for dissemination, we refer to the latter as application payload.

MPL transmits control and data messages. MPL data messages are used to disseminate the application payload between MPL forwarders. MPL control messages are used to inform neighbouring MPL forwarders about recently received MPL data messages.

MPL extends the Trickle algorithm with one more parameter, called the expiration counter (TIMER_EXPIRATIONS). It is used to stop the Trickle timer after TIMER_EXPIRATIONS timer events. The default value of the parameter is 3 and 10, for Trickle timers for MPL data and control messages, respectively.

MPL defines two forwarding strategies: proactive and reactive. Proactive forwarding relies only on MPL data messages for dissemination: each MPL forwarder upon receipt of a new MPL data message, attempts to transmit the message to its neighbours. The transmission is done based on a Trickle timer. After a pre-defined number of attempts, the forwarder will stop the Trickle timer and discard the message. Reactive forwarding uses MPL control messages to signal when MPL data messages need to be sent. Each MPL forwarder transmits MPL control messages based on a Trickle timer. Whenever an MPL forwarder receives an MPL control message, and from it learns that it has an MPL data message that is new to its neighbours, the MPL forwarder schedules the transmission of the given MPL data message based on a Trickle timer.

It is clear that the consistency model varies between the two schemes, and is significantly different to the RPL one. In the proactive strategy,

the reception of an unseen MPL data message is considered an inconsistency, while a seen MPL data message is considered as consistent. In the reactive forwarding strategy, an inconsistency is defined when the reception of an MPL control message indicates that at least one MPL data message is new to the sender or the receiver.

Due to the lack of openly available implementations of MPL, we are not able to evaluate the actual MPL draft. Therefore, we focus on the original Trickle algorithm as the forwarding strategy, which is very similar to the proactive forwarding policy in MPL. Therefore, the work presented in Section 5.5.1 applies to proactive MPL forwarding.

5.3 Related Work

The Trickle algorithm has been initially designed as an efficient method to disseminate software updates in LLNs [LPCS04]. However, since it only specifies *when* messages should be sent, and not *how*, it has been accommodated in many other protocols [Lev+08], such as network reprogramming [LL08], routing [Gna+09; RFC6550] and data dissemination [JVVG14].

Due to the widespread usage, various aspects of the Trickle algorithm have been extensively studied so far. Therefore, we categorize related work on Trickle based on the application area of the protocol where Trickle is used - data dissemination and routing (RPL).

5.3.1 Trickle as a data dissemination mechanism

The initial publications of Trickle [Lev+08; LPCS04] demonstrate that the algorithm scales well with network density, and is efficient in limiting redundant traffic. The first hypotheses was later proved in [MBBD14], by showing that the number of messages transmitted by the Trickle algorithm scales linearly in $k/I_{\max}$.

Several analytical models for various aspects of Trickle dissemination have been proposed. In [MBBD14], a descriptive model is presented, which shows how the message count and transmission rate depend on various Trickle parameters. The study gives approximations on the number of expected messages in a grid network topology. Another analytical model, described in [KGA12], focuses on random topologies,

and estimates the message count using the redundancy constant and average node degree as parameters.

Models for dissemination time for the Trickle algorithm have been developed in [BKG11; MBBD15]. Both models focus on line topologies, and extend the results to grid topologies. The models either do not take the redundancy constant in consideration, or assume that $k = 1$, to reduce the complexity of the analysis.

Early drafts of the MPL protocol have been simulated in [CVY13; LVHD14; OPT13]. The general conclusion of all three studies is that the performance of MPL heavily depends on the chosen Trickle parameters. Simply using the default parameters often results in sub-optimal performance, and other dissemination protocols easily outperform it. Proper parameters settings, set with the application protocol and the network topology in mind, give much better results and achieve Trickle's initial goals - fast dissemination time and little redundant traffic. However, even though all three studies explore a fairly broad range of values for the Trickle parameters, they fail to analyse the causes of poor performance in depth.

5.3.2 Trickle as a part of RPL

The RPL protocol uses the Trickle algorithm for transmission of DIO messages, which are essential in the construction of the routing overlay. Therefore, many studies on the performance of Trickle when applied to the RPL protocol haven been conducted.

Simulation results for the early versions of the RPL protocol are presented in [TOV10; Ko+11]. However, these versions of the RPL protocol do not use Trickle's suppression mechanism ($k = \infty$). Both works conclude that in some scenarios the performance of RPL is lacking and additional studies are needed for its usage in large-scale networks.

Later, Trickle's suppression mechanism was deemed necessary to ensure scalability of the protocol and has been made part of RPL's current RFC. However, there is no clear consensus on the recommended value for the redundancy constant. The Trickle RFC specifies that in protocols that want to avoid suppression, a high value of k (5 or 10) should be preferred to an infinite value. However, it also suggests that experimentally, values of 1-5 have been shown to provide good performance. Currently, the RPL RFC recommends using $k = 10$, which, according to the RFC's authors,

should be a conservative value. Another RFC [LG13] reports that values between 3 and 5 have shown good experimental results for RPL. To add to the confusion, as previously stated, the Collection Tree Protocol (CTP) does not use suppression at all ($k = \infty$), while MPL recommends using $k = 1$.

Based on the latest RPL RFC, additional simulation studies have been performed. Performance analysis using application level metrics is given in [AGBC11; ITN13]. However, these studies use the default Trickle parameters for RPL, and do not consider how these parameters can influence the chosen metrics. An extensive simulation study on the effect of the redundancy constant k and $I_{\min}$ on RPL's performance is given in [KG14]. In their study they consider random spatial networks with different densities and vary k between 1 and 15 and $I_{\min}$ between 4ms and 16ms. They observe that both the redundancy constant and $I_{\min}$ provide a trade-off between network convergence time and overhead control traffic (DIO's). However, the impact of $I_{\min}$ is limited during the convergence phase, while the redundance constant has more significant influence long-term and when the size of the network increases.

Recently, two studies identified scenarios where the Trickle algorithm exhibits unfairness in terms of load distribution [EG14; VM13]. The first study notices that when new nodes join a Trickle network, they immediately lose synchronicity to the rest of the network, and rarely get the chance to transmit. As a solution, they propose a re-synchronization method as part of Trickle. The second study observes that the unfairness can occur if the redundancy constant k is not configured properly with respect to the network density. Both studies conclude that the unfairness can lead to sub-optimal routes in RPL. The second study simulates the behaviour of RPL in grid and random networks, and shows that due to the bad routes, the unfairness ultimately leads to increased delays and increased energy usage. As a solution, they propose a modification of Trickle, which forces nodes to broadcast if they have been suppressed for a long period of time.

Finally, link instability was identified as a problem for new nodes in a network [Anc+14]. Due to the lack of link quality measurements, new nodes have been observed to blindly connect to the first available node in an RPL network, even though better alternatives might exist. They address this issue by adding a probing phase, where nodes first measure the link quality to their neighbors based on a Trickle timer,

before selecting a preferred parent. As a result, nodes take more time to join a network, but benefit from having more stable routes.

5.4 Impact of the redundancy constant

Clearly, the redundancy constant k is one of the most important parameters of the Trickle algorithm, but setting it properly is not trivial. As shown in the related work, optimally setting k depends greatly on the network topology and the application for which the Trickle algorithm is used. In this section, we focus only on the network topology and use RPL as a driver.

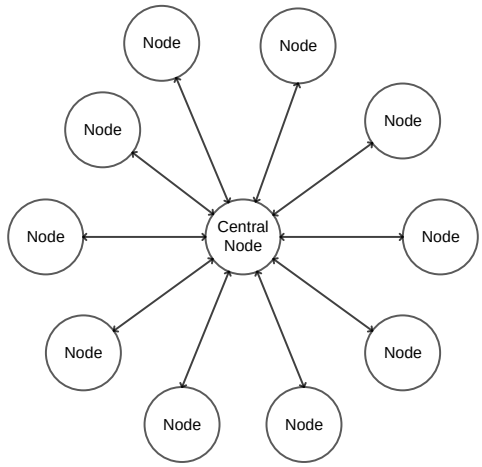


Figure 5.3: Unfair load distribution in a star network. If all nodes have the same value for the redundancy constant, the central node has smaller chance of transmitting compared to all other nodes.

As a rule of thumb, in sparse networks, where nodes have few neighbors, k should be set to a low value. Otherwise, if k is greater than the node degree, the suppression mechanism is practically disabled. Similarly, in very dense networks, a low value of k will suppress most nodes in the network. Such behaviour can lead to slower coverage of the network, particularly if bottleneck nodes are frequently suppressed. Therefore, a higher value of k is more reasonable, since it will give chance to more nodes to fight for the medium.

However, if a network is of mixed density, as is common in practice, having the same value of the redundancy constant will definitely lead to

sub-optimal performance. For example, in a synchronized star network of $n+1$ nodes, the central node has to compete with n nodes to transmit, while each other node has to compete with just one node (Figure 5.3). The probability for the central node to transmit is $min(\frac{k}{n+1}, 1)$, while all other nodes transmit with probability $\begin{cases} \frac{n}{n+1}, k = 1 \\ 1, k > 1 \end{cases}$, which is unfair.

5.4.1 *Adaptive-k*: a density-aware redundancy constant

With these considerations in mind, it makes sense to set k for each node individually. Since this is a task one would like to prevent from having to do manually, ideally we would want nodes to set their own k in a distributed fashion. We know nodes keep track of the number of Trickle messages they receive during an interval with a counter c. This counter c contains implicit information on the number of neighbors of that node, i.e. the node's degree, which is a dimensionless number that gives an estimate on the local density of the network. Therefore, we propose an extension to the Trickle algorithm, called *adaptive-k*, which leverages this information and allows nodes to set their value of k autonomously. This extension is done by a slight modification of rule 5 of the algorithm:

> 5^*. When a node's timer reaches time I, it sets k equal to $f(c)$, increases its interval length to $\min(2I, I_{\max})$ and starts a new interval.

Here f is some predefined function, which is the same for all the nodes of the network. We argue that a natural candidate is the following function:

$$f(c) = \begin{cases} k_{\min}, & \alpha c < k_{\min}, \\ \lfloor \alpha c \rfloor, & k_{\min} \leq \alpha c \leq k_{\max}, \\ k_{\max}, & \alpha c > k_{\max}, \end{cases} \tag{5.1}$$

with some fixed $\alpha \in [0, 1]$ and $k_{\min}, k_{\max} \in \mathbb{N}$. The function f should be bounded by below by some $k_{\min}$ to avoid a deadlock with all nodes having $k = 0$. One should think of $k_{\min}$ being small, i.e. 1 or 2. Throughout this paper we assume $k_{\min} = 1$. Furthermore, the function f should be bounded from above by $k_{\max}$ to ensure scalability of the algorithm. In

line with the recommendations in the Trickle RFC, $k_{\max} \cdot I_{\min}$ should at least be two to three times as long as it takes to transmit $k_{\max}$ packets.

Intuitively, this extension does what we would like it to do. Whenever a node receives many broadcasts during an interval, it knows it has many neighbors, and hence it should increase k in order to be able to compete for the medium. When a node receives few transmissions, it either does not have a lot of neighbors, or its neighbors are having a hard time broadcasting their own information, and for both cases the node should lower its redundancy constant k.

Note that our extension is backward compatible with the Trickle RFC: the RFC itself acknowledges that nodes can be configured with a different redundancy constant, with the possible drawback of an uneven load distribution. In the next section we show that Trickle with *adaptive-k* actually leads to a more even load distribution.

5.4.2 Evaluation of the adaptive redundancy constant

A more detailed analytical analysis on the performance of Trickle with *adaptive-k* is done in [MSL15]. Here, we present the main results from that study, for three different network topologies. In the analysis we assume that there is no message loss and there is no delay in transmission. Furthermore, we assume all nodes to have the same data and that $I = I_{\max}$ for all nodes.

5.4.2.1 Single-hop network

Consider a single-hop network of n nodes with the original Trickle algorithm. Then, according to [MBBD14], the maximum number of transmissions per interval will be either k or $2k$, depending on whether the nodes are synchronized or unsynchronized, respectively ($k \leq n$).

However, if Trickle with *adaptive-k* is used, the number of transmissions depends on α. If $\alpha < 1$ and nodes are synchronized, then in the first interval there will be $k_{\max}$ transmissions. Then, in each following interval, all nodes will gradually decrease their redundancy constant k until it finally reaches $k = k_{\min}$. From then on, maximum number of transmissions per interval will be $k_{\min}$.

Similar behaviour is expected if nodes are not synchronized and $\alpha \leq 1/2$. In the first interval there will be at most $2k_{\max}$ transmissions. Then, in each following interval, all nodes will gradually decrease their redundancy constant k, until it finally reaches $k = k_{\min}$. From the on, maximum number of transmissions per interval will be at most $2k_{\min}$.

Finally, if $\alpha > 1$ for synchronized nodes, or $\alpha > 1/2$ of unsynchronized nodes, then all nodes will gradually increase their redundancy constant k until it reaches $k = k_{\max}$. In any subsequent interval, the maximum number of transmissions per interval will be either $k_{\max}$ or $2k_{\max}$, depending on the synchronicity of the nodes.

From this case we can conclude that in the worst case, Trickle with *adaptive-k* will have as many transmissions per interval as the unmodified Trickle algorithm with $k = k_{\max}$. Furthermore, this shows that *adaptive-k* has to be bounded, or the suppression mechanism will stop working properly if $\alpha > 1$ and n grows.

5.4.2.2 Star network

We previously showed that in a star network of $n + 1$ nodes as in Figure 5.3, the unmodified Trickle algorithm is unfair, starving the central node from transmissions. If Trickle with *adaptive-k* is used in the same topology, then if $\alpha \leq 1$, every non-central node will set $k = 1$ immediately after the first interval, while the central node will keep adapting its redundancy constant k according to a Markov chain (Appendix from [MSL15]). If $n \to \infty$, then the probability that the central node broadcasts is:

$$p_\alpha = 1 - \left(\sum_{i=0}^{\infty} \frac{\alpha^{i(i+1)/2}}{i!} \right)^{-1}. \tag{5.2}$$

On the other hand, the probability that a non-central node broadcasts is:

$$p_\alpha^* = \frac{1}{\alpha}(1 - p_\alpha). \tag{5.3}$$

Finally, if $\alpha = 1$, the Trickle algorithm with *adaptive-k* is asymptotically fair, even though the central node has to compete with infinitely many other nodes:

$$p_1^* = p_1 = 1 - \frac{1}{e}. \tag{5.4}$$

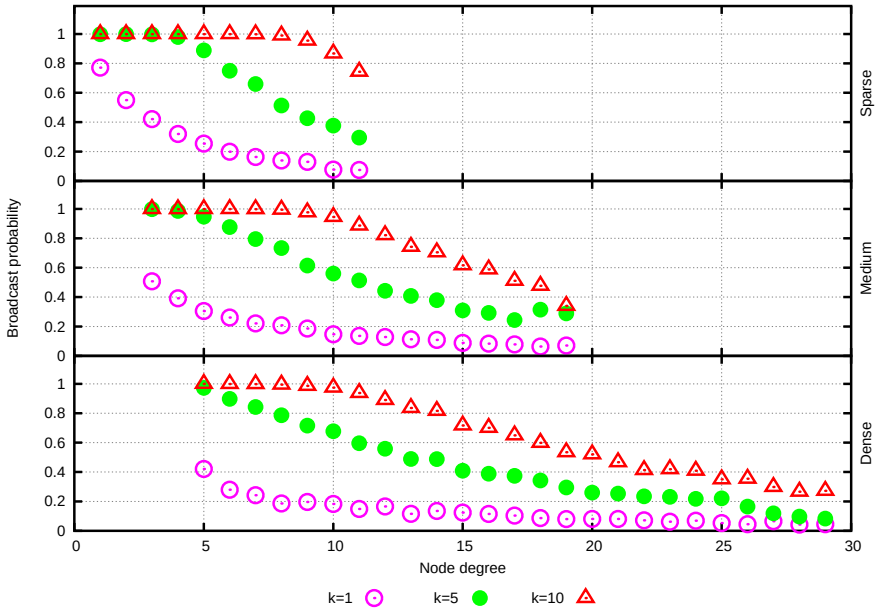


Figure 5.4: Simulated broadcasting probability per node degree for the *original* Trickle algorithm. The three figures correspond to the three different network densities.

Furthermore, since each transmission of the central node will suppress the transmission of all other nodes in the same interval, Trickle with *adaptive-k* will transmit fewer broadcasts per interval than the original Trickle algorithm.

5.4.2.3 Random spatial network

To verify the performance of Trickle with *adaptive-k* in a more realistic setting, in [MSL15] several simulations are performed on different network topologies. The simulations cover the original Trickle algorithm with $k = 1, 5$ and 10, as well as the Trickle algorithm with *adaptive-k* for four different functions $f(c)$ of the form as in Equation (5.1), with several values of α in the range $[\frac{1}{2}, 1]$. The rest of the parameters are set to $k_{\min} = 1$ and $k_{\max} = 30$.

The simulations are done over three classes of random spatial networks of 200 nodes placed uniformly in a square region. The three types of topologies have an average node degree of 5, 10 and 15, representing

sparse, medium and dense networks. For each setting and density, 10 networks with different topologies are simulated for 200 time units, where all nodes start with $I = I_{\max} = 1$ and uptodate information. The main calculated metric is the average broadcasting probability of every node degree, i.e., the fraction of intervals nodes with that number of neighbors broadcast.

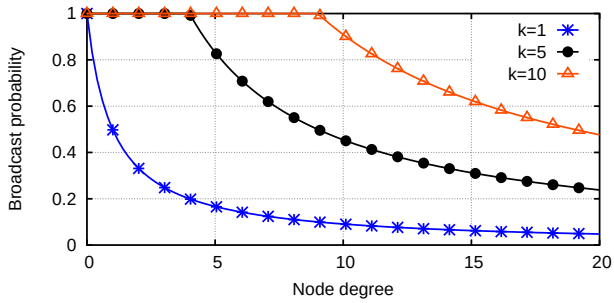


Figure 5.5: Estimate for the broadcasting probabilities for different values of k.

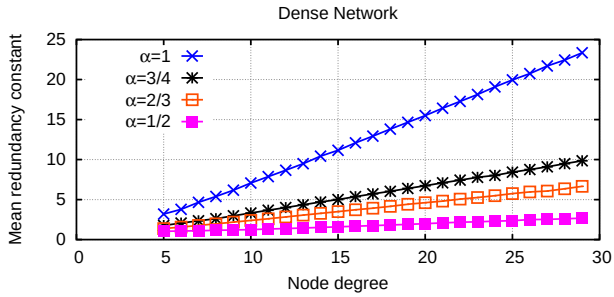


Figure 5.6: The average redundancy constant per node degree.

Figure 5.4 shows the simulation results for the original Trickle algorithm with different values of the redundancy constant k. As expected, nodes with less that k neighbours broadcast in virtually every interval, while nodes with more neighbours broadcast with a much lower probability. Roughly, the probability for a node with N neighbours to broadcast in an interval is $\min(1, k/(N+1))$, plotted in Figure 5.5. The same trends can be seen for all network densities, with all values of the redundancy constant k.

Figure 5.7 shows the simulation results for the Trickle algorithm with *adaptive-k* with different values of the parameter α. The average value of the redundancy constant k per node degree is shown in Figure 5.6. From

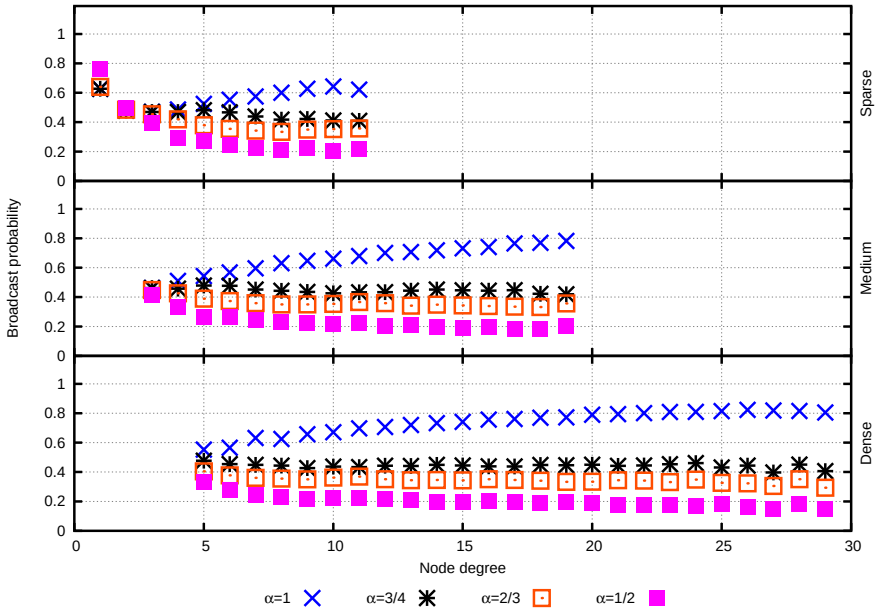


Figure 5.7: Simulated broadcasting probability per node degree for the *adaptive-k* Trickle algorithm. The three figures correspond to the three different network densities.

these results, it is evident that the algorithm works as intended, linearly increasing the value of k depending on the number of neighbours a node has. As a result, we can observe that the transmission load is distributed more fairly compared to the original Trickle algorithm. However, it still depends greatly on the choice of α. With $\alpha = 1$, there is a clear tendency to favor high degree nodes, while $\alpha = 1/2$ shows a slight tendency to favor low degree nodes. From these simulations, the most appropriate setting for α seems to be between $2/3$ and $3/4$. These two settings show the best trade off in terms of communication load and fairness in transmission load.

5.4.3 RPL Evaluation

To confirm that the *adaptive-k* extension to the Trickle algorithm can improve performance, we resort to simulations and experimental evaluation. We use the RPL protocol as a case study, since previous studies have shown that unfair load distribution can lead to problems in RPL. In order

to be able to forward messages as efficiently as possible, nodes need to quickly update their rank to the *optimal* rank. However, as we saw in the previous section, the original Trickle algorithm tends to favor nodes with low degree. Therefore, low degree nodes will be able to broadcast DIO's more often than high degree nodes. This introduces the problem that nodes tend to favor low degree nodes as their parents, possibly leading to sub-optimal routes, since routes through high degree nodes might be more efficient. One hopes that using Trickle with *adaptive-k* could solve this problem, since it distributes the transmission load more evenly.

5.4.3.1 Simulation results

We implement *adaptive-k* as part of the Contiki 2.7 operating system [DGV04]. Contiki includes a full IPv6 over LoWPAN (6LoWPAN) stack, together with an implementation of the RPL protocol, called ContikiRPL. We simulate different topologies in Cooja, a cross-level simulator for Contiki [Ost+06]. Cooja internally uses the MSPsim device emulator for cycle accurate Tmote Sky emulation, as well as a symbol accurate emulation of the CC2420 radio chip. We use the Unit Disk Graph Radio Medium (UDGM) model for radio propagation, with no loss. UDGM penalizes collisions heavily, while end-to-end delivery fails only due to filled MAC queues. At the link layer, we use the default Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) implementation in Contiki with no RDC (nullrdc). The RPL DODAG is formed according to the Objective Function Zero (OF0) [RFC6552]. OF0 in Contiki is implemented as a hop-count based selection metric, which uses local Expected Transmission Count (ETX) measurements to select between parents with the same rank. Unnecessary parent switches are avoided by adding a simple hysteresis mechanism [RFC6719].

We look at topologies with three different network densities, where one root and 100 non-root nodes are uniformly placed in a square area of 100×100 meters. For each topology, named sparse, medium and dense, the transmission range is such that the average node degree is 5, 10 and 15, respectively. We simulate a Constant Bit Rate data gathering application - every non-root node sends one 80-byte packet (including all headers) to the root node every minute. For each topology, we consider the original Trickle algorithm with $k = 1, 5$ and 10 and *adaptive-k* Trickle with $\alpha = \frac{2}{3}$, $k_{\min} = 1$ and $k_{\max} = 10$. We simulate each configuration 100 times, where each simulation runs for 2 hours with $I_{min} = 2^3 ms$

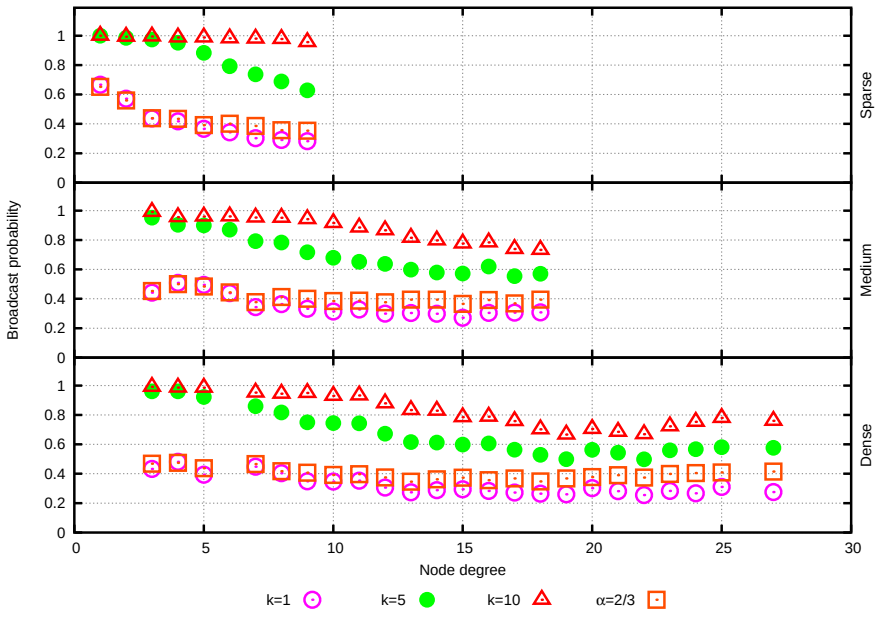


Figure 5.8: Average DIO broadcasting probability per node degree for the Trickle algorithm with different values for k and the *adaptive-k* Trickle algorithm. The three figures correspond to the three different network densities.

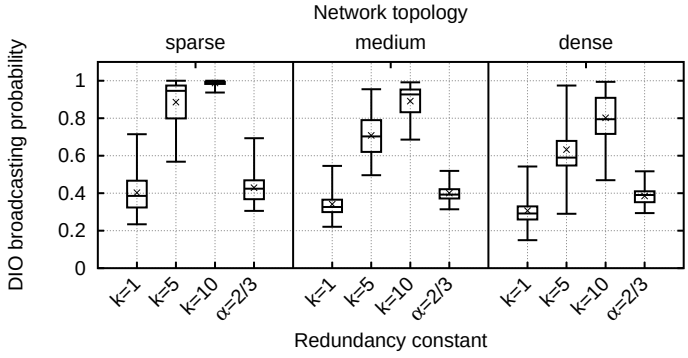


Figure 5.9: Cumulative statistics on the average DIO broadcasting probability per node for different network densities. The whiskers show the minimum/-maximum value, the start and end of the box show the 25th/75th percentile, the line in the box is the median, and the cross is the mean.

and $I_{\max} = 2^{23}ms$. We measure the mean time until formation of the first DODAG, the mean number of DIO transmissions per node and the mean network stretch after 2 hours. Network stretch is defined as the fraction of nodes that have a rank higher than the minimal rank, i.e. take more hops to reach the root node than through the optimal path.

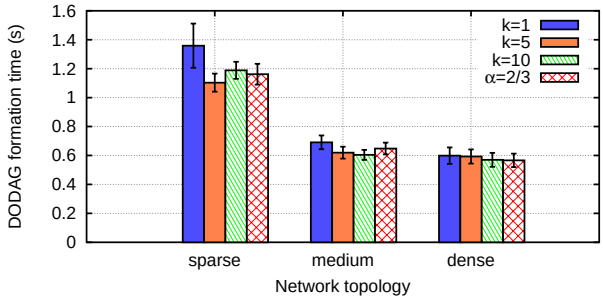
First of all, as expected, we find that with the original Trickle algorithm, the chance that a node broadcasts a DIO reduces as the network density increases (Figure 5.8). In a sparse network, a high redundancy constant disables suppression, while in a dense network nodes have less and less chance of transmission. On the other hand, Trickle with *adaptive-k* scales well with network density, and provides almost equal DIO broadcasting probability irrespective of the network density. The cumulative statistics in Figure 5.9 confirm these findings.

Next, we analyse the RPL-level metrics. Firstly, we find that for the simulated topologies, the DODAG formation time is not greatly affected by the choice of k; only for sparse networks does the formation time suffer from low values of k (Figure 5.10a). Secondly, for the original Trickle algorithm, the average number of DIO's increases quickly with k, as expected. However, the DIO count of Trickle with *adaptive-k* is comparable to that of the original algorithm with $k = 1$ (Figure 5.10b). Lastly, for the original Trickle algorithm with low k, the average network stretch remains high, probably due to Trickle favoring low degree nodes. As k increases the network stretch decreases; nodes are able to broadcast more easily, allowing for discovery of better routes at the cost of high overhead. However, we find that for *adaptive-k*, for every scenario the network stretch after 2 hours is almost zero; only in a few cases there are 2 or 3 nodes that have not yet discovered the optimal route (Figure 5.10c), which can be avoided by increasing $k_{\min}$. In summary, the network stretch shows that Trickle with *adaptive-k* allows for good routes to be discovered, as if k was high, while only broadcasting few DIO's, as if k was set low.

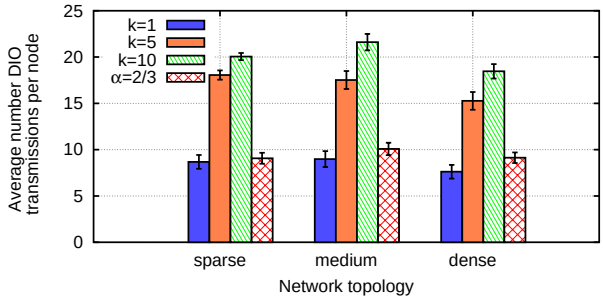
5.4.3.2 Experimental results

To confirm the simulation results, we implement the same code as in the previous section, on a set of 43 WSN430 nodes in the Rennes FIT IoT-Lab physical test bed [ii]. WSN430 nodes have the same MSP430 microcontroller and TI CC2420 radio chip as the Tmote Sky. We configure

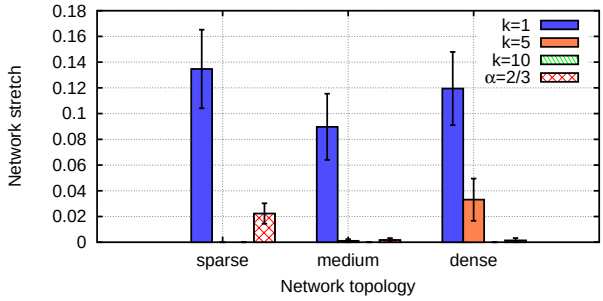
[ii] http://www.iot-lab.info



(a) DODAG formation time



(b) Average number of DIO's



(c) Network stretch

Figure 5.10: Influence of the redundancy constant in RPL for different topologies. All values show the average of 100 runs and the 95% confidence interval.

the nodes to use the minimum available transmission power (-25 dBm), which is enough to form a network with at most 4-hops to the root. To improve stability and enable more accurate measurement, we use larger Trickle intervals, $I_{min} = 2^4 ms$ and $I_{\max} = 2^{24} ms$. The rest of the parameter settings are identical as in the simulations. For each value

of the redundancy constant, we perform five experiments, where each experiment runs for one hour. As nodes are booting up randomly, links are fairly unstable, and there is no central clock in the system, we only show the results for the overall number of transmissions as measured at the link layer, and the end-to-end packet delivery ratio. All charts show the average measured values and the 95% confidence interval of the mean.

The experimental results confirm the simulation results; setting a proper value of the redundancy constant significantly impacts the overall traffic in the network (Figure 5.11). In this particular network topology, a high value for the redundancy constant does not improve routing, and therefore should be set low. The proposed *adaptive-k* extension handles the situation well, balancing between the overhead control traffic and high end-to-end packet delivery ratio.

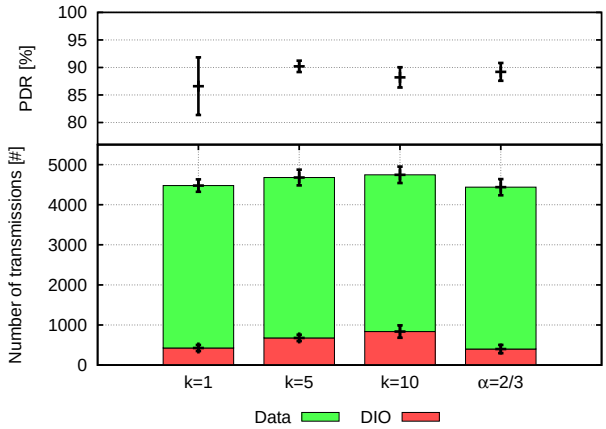


Figure 5.11: Total number of DIO transmissions, data transmissions and end-to-end packet delivery ratio (PDR) during 1h of operation on 43 nodes at the IoT-Lab test bed.

5.4.4 Summary

The redundancy constant is an important factor for the performance of the Trickle algorithm. The current Trickle standard proposes a fixed value of the redundancy constant, which is sub-optimal in networks of varying density. As we demonstrated, a fixed value for the redundancy constant makes the Trickle algorithm unfair in terms of load distribution, favoring

nodes with few neighbors. Moreover, depending on the application where Trickle is used, this unfairness can lead to various consequences, such as creation of sub-optimal routes (RPL).

The *adaptive-k* extension to the Trickle algorithm makes the algorithm more robust to network topology. As the results show, it is able to scale well with varying network density. The extension can be further tweaked to trade-off between redundancy and overhead via the α parameter.

5.5 Lower layer interference on Trickle operation

All of the previously mentioned protocols that use the Trickle algorithm operate either at the network or the application layer. The Trickle algorithm essentially controls the generation of packets within these protocols, while lower layers (i.e. link layer and physical layer) are responsible for the actual transmission of the data packets (Figure 5.12). Now, we focus on the interplay between Trickle and the protocols at the link layer. We first give an overview of the link layer of low power radios, and then show how its behaviour can influence Trickle operation.

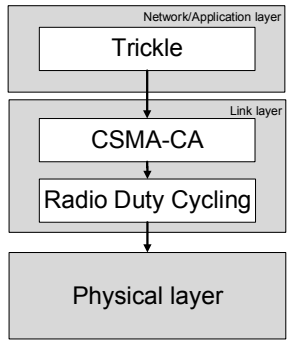


Figure 5.12: Flow of Trickle packets in the Contiki operating system [DGV04].

5.5.1 Low-power link layer protocols

The link layer of IEEE 802.15.4 [IEEE15.4e] low-power radios is built of two components - a MAC protocol and a radio handling protocol. The MAC protocol handles the allocation of the shared medium among nodes and covers retransmissions in case of collisions or packet loss. The

radio handling protocol determines the efficient use of the radio during the periods allocated by the MAC protocol. We will now give a detailed description of both protocols.

5.5.1.1 CSMA/CA protocol

The link layer has the responsibility to deliver messages between devices in the same Local Area Networks (LANs). It can also include means to repair imperfections of the physical layer (ex. detect transmission errors), multiplex access between multiple devices on a shared media, detect simultaneous access attempts and recover from them. For the last three tasks, wireless networks often use the CSMA/CA protocol. It is a probabilistic protocol, where each node first verifies that there are no ongoing transmissions in the area, and then attempts to transmit.

The IEEE 802.15.4 MAC defines two flavors of the CSMA/CA protocol, depending on the operational mode in use: slotted CSMA/CA, used in beacon-enabled mode, where beacons are sent to synchronize nodes to a super-frame structure; and unslotted CSMA/CA, used in non beacon-enabled mode, where no beacons are sent out and there is no synchronization between nodes. In this work, we focus on unslotted CSMA/CA, but the same concepts apply to slotted CSMA/CA as well.

In unslotted CSMA/CA, the basic time unit is the back-off period BP, which is related to the transmission time of a frame. Every node maintains two variables for each frame it wants to send: a back-off exponent BE, and a counter for the number of back-offs for the current transmission NB. These variables are controlled by three parameters: the minimum back-off exponent $BE_{\min}$, the maximum back-off exponent $BE_{\max}$ and the maximum number of back-offs $NB_{\max}$.

The flowchart of the algorithm is shown in Figure 5.13. Initially, $NB = 0$ and $BE = BE_{\min}$. Before each transmission, each node first waits for a random number of BPs ranging from 0 to $2^{BE} - 1$. After the initial back-off, the node performs a Clear-Channel Assessment (CCA) to determine whether the channel is free. If the channel is free, the node proceeds with the transmission. Otherwise, it increases NB by one, and sets BE to $\min(BE + 1, BE_{\max})$. If $NB \leq NB_{\max}$, the entire procedure is repeated. After $NB_{\max} + 1$ failed attempts, the frame is dropped from the MAC queue.

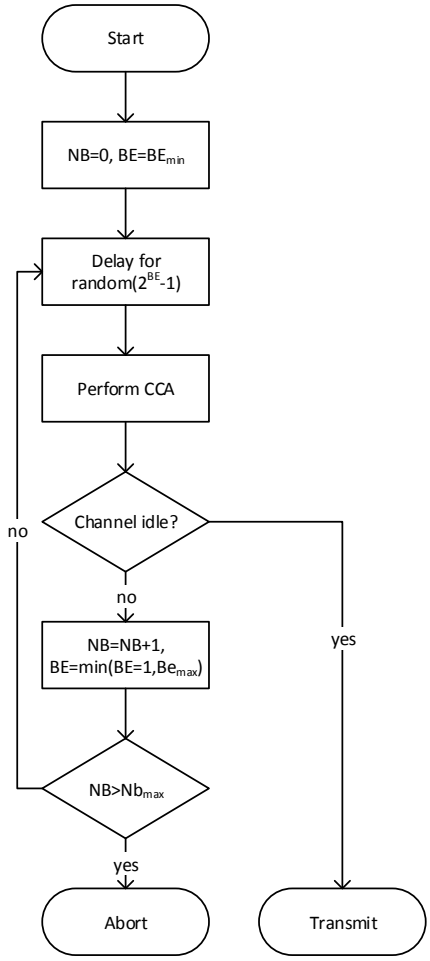


Figure 5.13: Flowchart of the CSMA/CA protocol.

5.5.1.2 Radio duty cycling

The MAC layer of low-power radios often includes a second component next to the CSMA/CA protocol - the radio handling protocol. Radio transceivers are among the biggest sources of energy consumption in low-power wireless devices. Therefore, low-power wireless devices must trade-off between keeping the radio transceiver off, to save energy, and periodically waking up to be able to receive data from their neighbors.

During the years, many RDC protocols have been proposed. They can be categorized into synchronous, where nodes are synchronized with their neighbouring nodes (eg. S-MAC [YHE02], T-MAC [DL03]), and asynchronous, where no pre-synchronization is required. Asynchronous RDCs can be further categorized into sender initiated (eg. Low Power Listening [HC02], X-MAC [BYAH06], ContikiMAC [Dun11b]) and receiver initiated protocols (eg. Low Power Probing [MELT08], A-MAC [Dut+10]). Sender initiated RDC protocols give the transmission incentive to the senders: senders wake up receivers to receive a transmission. Receiver initiated protocols give the incentive to the receivers: receivers inform senders when they are prepared to receive a transmission. Finally, hybrid approaches have been developed, which combine features from any of the given categories (eg. WiseMAC [EHD04]).

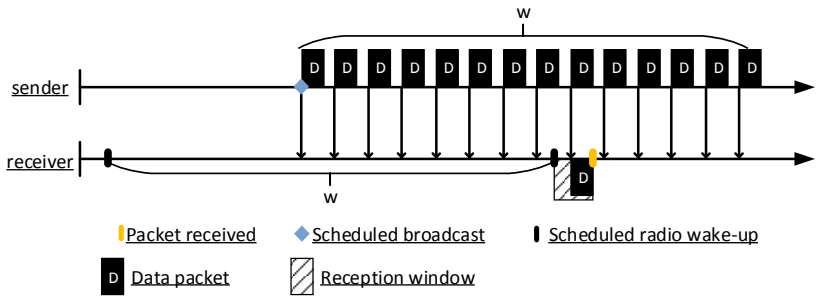


Figure 5.14: ContikiMAC broadcast transmission. In ContikiMAC, broadcast transmissions are sent with repeated frames for the full wake-up interval. This illustration is reproduced based on [Dun11b].

In this work, we consider ContikiMAC [Dun11b], a sender initiated RDC. It is similar to the Coordinated Sampled Listening (CSL) protocol, introduced in the IEEE 802.15.4e standard. A brief description of ContikiMAC follows.

By default, every node has its radio turned off. Periodically, at regular intervals of w time units, each node turns its radio on to check for incoming traffic. If a transmission is detected, the radio is kept on until the frame is received. Transmissions are non-periodic, originating from the upper layer(s). When they arrive, a CCA is done to see if the medium is free. If it is free, the node starts transmitting immediately. Broadcast transmissions should be received by all nodes, irrespective of their wake

up intervals. Therefore, a broadcast transmission will always be repeated for w time units (Figure 5.14), so that each node will at least once turn on its radio during the transmission. Hence, assuming an idle channel, the worst-case latency as defined in the Trickle RFC, is w. However, this makes broadcasts expensive both in terms of delay and consumed energy.

The main configuration parameter for ContikiMAC is the radio wake-up frequency $1/w$, i.e. how often each node samples the radio. This parameter also dictates the maximum duration for each individual transmission w. Typically, the wake-up frequencies are set to 4Hz, 8Hz or 16Hz, giving wake up intervals of 250ms, 125ms and 62.5ms, respectively.

5.5.2 Interference scenario

A common feature of almost all link layer protocols is that transmissions are not instantaneous, and there is a variable delay between the intent to start a transmission and the actual reception. For instance, CSMA/CA may add a random delay before each transmission to avoid collisions. In sender initiated RDC protocols as ContikiMAC, the transmission starts almost immediately after it is received from the upper layers, but it is not completed until the receiver performs its periodic wake up to sample the channel. Similarly, in receiver initiated RDC protocols, the transmission is delayed until the sender receives a request from the receiver, which is again periodically scheduled. Finally, in case of collisions, in both cases, CSMA/CA will re-schedule transmissions after a certain back-off period. The delayed completion of a transmission creates a window where upper layer protocols may think that a transmission has been completed, while in fact, it is not. This causes unintended and inefficient messaging, as the transmission delay and retransmissions may move from one to another Trickle interval.

For example, consider a network consisting of two nodes (Figure 5.15). Both nodes use unslotted CSMA/CA in combination with RDC at the MAC layer. Packet transmission is regulated by the Trickle algorithm ($k = 1$, $\eta = 1/2$). Both nodes start a Trickle process at the same time, with consistent information for dissemination. They choose transmission times t_1 and t_2, respectively, such that $t_1 < t_2$. Both counters are initially set to zero ($c_1 = c_2 = 0$). At time t_1, since $c_1 < k$, node 1 sends a packet to its MAC layer. Then, it does a successful CCA and starts transmitting the packet. Node 2 has its next wake-up scheduled at time $t_r > t_2$.

Consequently, at time t_2 node 2 has not yet received node 1's broadcast and will decide to transmit itself, sending a Trickle packet to its MAC layer. Since at this time the channel is busy, CSMA/CA will delay this transmission until $t_2 + bo$, where bo is the back-off time. At time t_r, node 2 receives the transmission from node 1, setting $c_2 = 1$, making the queued packet in the MAC layer obsolete. However, since there is no link between the MAC queue and the application layer, the packet will be sent at $t_2 + bo$. This effect can be cascaded if multiple nodes exhibit the same behavior. Moreover, it is possible that node 2's broadcast is delayed into its next Trickle interval (Figure 5.15), causing node 1 to suppress its next broadcast, further disrupting the Trickle process.

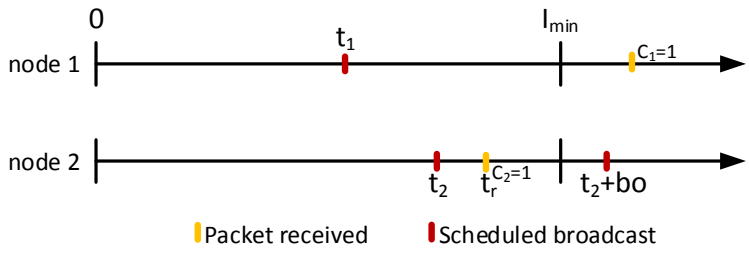


Figure 5.15: Link layer interference on Trickle timing. Nodes 1 and 2 get updated at the same time, and they select transmission times at t_1 and t_2, respectively. If the reception for node 2 (t_r) is scheduled to be after t_2, node 2 will queue a Trickle packet at t_2, even though there is a packet in the air from node 1. Due to CSMA/CA, this packet will be transmitted after the back-off, at time $t_2 + bo$.

5.5.2.1 Case study: CSMA/CA and ContikiMAC

We will now use the Contiki operating system for a case study on the impact of MAC interference on Trickle timing. Contiki 2.7 utilizes the ContikiMAC RDC protocol with a radio wake-up interval length of w, together with a slightly modified version of the unslotted CSMA/CA protocol. Firstly, the default parameters $BE_{\min} = 0$, $BE_{\max} = 3$ and $NB_{\max} = 3$, force CSMA/CA to skip the first back-off. Secondly, the back-off period is equal to the length of the wake-up interval of ContikiMAC ($BP = w$). As w is the worst-case transmission time for ContikiMAC, this ensures that any retransmissions are attempted after the current

transmission has finished. Thirdly, the CCA check is delegated to the RDC layer. Finally, the back-off exponent BE is increased only when no acknowledgment is received for sent unicast frames. Since Trickle-based data dissemination uses only broadcast packets, for which no acknowledgment is needed, a back-off can only occur due to a failed CCA or a detected collision. In both cases, BE remains one, causing the back-off for broadcasts to remain $BP = w$.

5.5.2.2 Scenario 1: Single-hop network

We now analyze the likelihood that the scenario discussed at the beginning of this section occurs under ContikiMAC. Denote by $\mathbb{P}_2^{\text{bo}}$ the probability that a CSMA/CA back-off takes place in a network of two nodes. For simplicity, we assume the nodes to be synchronized, which would be the case if they got updated simultaneously. We assume that packets are received at radio wake-up and $I_{\min} = m \cdot w$, where $m \geq 2$ is a constant and w is the radio wake-up interval. We require $m \geq 2$, since otherwise a node will never be able to finish a transmission within the same Trickle interval as it was scheduled. Furthermore, assume that the Trickle process has $k = 1$ and $\eta = 1/2$. A CSMA/CA back-off will take place if either node 1 or 2 pick their transmission time during a broadcast of the other node and before their radio wake-up and reception. Hence, we can write:

$$\mathbb{P}_2^{\text{bo}} := 2\mathbb{P}[t_1 \leq t_2 \leq t_r \leq t_1 + w] = 2 \int_{I_{\min}/2}^{I_{\min}} \mathbb{P}\left[t_2 \in [t_1, t_r] \mid t_1 = t\right] \mathrm{d}\mathbb{P}[t_1 \leq t]. \tag{5.5}$$

Since both t_1 and t_2 are chosen uniformly in $[I_{\min}/2, I_{\min}]$ and a broadcast starting at time t is received by the non-transmitting node uniformly at $t_r \in [t, t + w]$, some calculus gives:

$$\mathbb{P}_2^{\text{bo}} = \frac{2}{m} - \frac{4}{3m^2}. \tag{5.6}$$

Note that this probability only depends on m, the ratio between the length of an interval $I_{\min}$ and the length of a broadcast w. For the MPL standard $I_{\min} = 10w$, this implies $\mathbb{P}_2^{\text{bo}} = 0.1925$, which is relatively large.

Extending these calculations and noting that nodes choose their timers independently, the probability that b CSMA/CA back-offs occur and $b+1$ transmissions are scheduled during an interval in a single-hop network consisting of n nodes is given by:

$$\mathbb{P}^{\text{bo}}_{n,b} := n\binom{n-1}{b}\mathbb{P}\left[t_2 \in [t_1, t_r]\right]^b \mathbb{P}\left[t_r \leq t_2\right]^{n-b-1}. \tag{5.7}$$

Like (5.5), this expression can be evaluated analytically and allows us to calculate the probability $\mathbb{P}^{\text{bo}}_n$ that at least one CSMA/CA back-off ($b > 0$) takes place during a single interval in a single-hop network consisting of n nodes:

$$\mathbb{P}^{\text{bo}}_n := 1 - \mathbb{P}^{\text{bo}}_{n,0} = 1 - \frac{1}{m^n}\left((m-1)^n + \frac{1}{2n-1}\right). \tag{5.8}$$

Moreover, calculating the expected number of redundant transmissions per interval due to poor interaction between Trickle and the CSMA/CA protocol gives:

$$\mathbb{E}[N^r_n] := \sum_{i=0}^{n-1} i\mathbb{P}^{\text{bo}}_{n,i} = \frac{n}{m} - \frac{1}{n+1}\left(\frac{2}{m}\right)^n. \tag{5.9}$$

Hence, the expected number of obsolete broadcasts per interval due to timing issues grows linearly with the size of the single-hop broadcast range. This is intuitive, since every node has the same probability of scheduling a back-off. If Trickle worked as designed, there would be only one packet per interval [iii].

5.5.2.3 Scenario 2: A bottleneck network

Consider now a network of four nodes, with connectivity as in Figure 5.16. This type of connectivity, where part of the network is reachable only through a single bridge node, is common, for example, in street lighting networks. Again all nodes use CSMA/CA in combination with ContikiMAC and run a Trickle dissemination process. The Trickle process has $k = 1$, $\eta = 1/2$ and $I_{\min} = m \cdot w$, where $m \geq 2$ is a given constant. Initially, all nodes have consistent information and $I = I_{\max}$.

Suppose at time 0 nodes 1 and 2 receive an update simultaneously from

[iii] For a complete calculation of Equations (5.5-5.9), see Appendix A of [SMCL15]

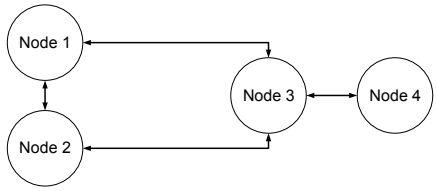


Figure 5.16: Example of a bottleneck network consisting of 4 nodes, where node 3 is the bottleneck.

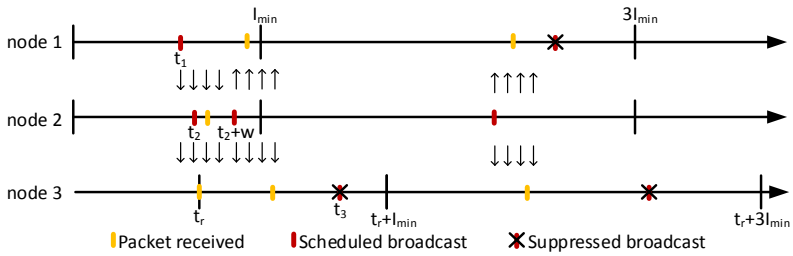


Figure 5.17: Suppression of Trickle updates due to link layer interference. Nodes 1 and 2 get updated at the same time, and select transmission times at t_1 and t_2, respectively, with the periodic channel check for node 2 (t_r) scheduled to be after t_2. Node 2 queues a Trickle packet at t_2. Due to busy media, CSMA/CA re-schedules the packet for $t_2 + w$. In the mean time, node 3 gets updated and starts a new Trickle interval. The re-transmission at $t_2 + w$ causes node 3 to suppress its transmission in the first interval (t_3). As node 1 and 2 started the second interval earlier than node 3, there is a high probability that they will suppress any future transmissions from node 3.

a close-by source, set $I = I_{\min}$ and start a new interval (Figure 5.17). Node 1 is the first node to schedule a broadcast, which it starts to transmit at time t_1. As we have seen in the previous scenario, node 2 will schedule a broadcast before receiving node 1's broadcast with probability $\mathbb{P}_2^{\text{bo}}$. If this happens, the MAC protocol will cause node 2 to delay its transmission until time $t_2 + w$. Before this time, however, node 3 will have been updated by node 1's transmission, and will start a new interval of length $I_{\min}$ and schedule a transmission at time t_3. Now node 2's transmission follows, suppressing node 3's transmission at time $t_3 > t_2 + w$ and consequently delaying the time that node 4 is updated. In its next interval, node 3 will broadcast only if it starts transmitting before it receives a broadcast by nodes 1 and 2. However,

due to the synchronization caused by the Trickle protocol, this has a small probability, as shown in Figure 5.17. The same problem occurs in the following intervals. Only when node 4 eventually transmits its old information, which depends on $I_{\max}$, it will reset node 3's Trickle process and an update will follow.

In general, if node 3 is connected with n synchronized nodes trying to update it, the previously described scenario occurs with probability $\mathbb{P}_n^{\text{bo}}$ (see equation (5.8)). We have plotted this probability and compared it with simulations for different values of m and n in Figure 5.18. From the plot it is clear that such an event is not rare. Given that such an event occurs, the probability that node 3 will ever broadcast in the following intervals before being suppressed by its neighbors is small, even for $n = 2$. Therefore, in such an event, with high probability node 4's update is delayed until it advertises its own old information, resetting the Trickle process of node 3. This gives an expected delay of approximately $\frac{1}{2}I_{\max} + \frac{3}{4}I_{\min}$, which is possibly very large since $I_{\max}$ is generally large. If node 4 has neighbors suppressing its own transmissions, then the expected delay will be even larger.

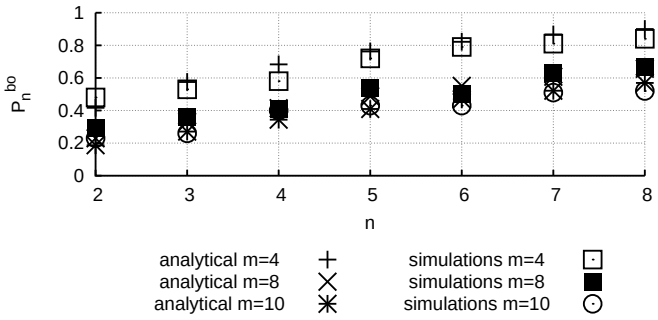


Figure 5.18: Analytical and simulation results of the probability that node 4 is updated after the second Trickle interval, for different values of m ($I_{\min} = m \cdot w$).

5.5.3 Cleansing MAC

In order to reduce the interference of the link layer on Trickle timing, we propose adding a *Cleansing* mechanism to the MAC layer. If Trickle is treated as a network primitive, as suggested in [Lev+08], known at both the network and link layer, then some decision making can be done at the link layer. Assuming that the MAC layer maintains separate queues

per destination, whenever a new Trickle packet arrives from the network, the Cleansing MAC will purge any queued outgoing Trickle packets. This will lead to less redundant packets in the network, and will minimize the bottleneck problem from the previous section.

In most cases, purging outgoing Trickle packets improves Trickle performance in terms of messaging and delay, and does not lead to functional incorrectness. It remains consistent with the software design of LLNs, as any purged packet can be seen as a message loss, and applications are already able to handle that situation. However, we can identify two scenarios where performance-wise, purging can be considered to be harmful.

The first scenario is when $k > 1$, a purged Trickle message might not be obsolete. However, this should have minimal impact on the network, since only a small fraction of messages within each single-hop broadcast domain will be purged. Moreover, other nodes in reach will make up for the purged transmission.

The second scenario is when a Trickle message with an old value arrives, and the Cleansing MAC protocol purges an outgoing Trickle message with a new value, increasing the overall propagation delay. However, the effect of the purge is limited, as due to the old message, the Trickle interval of the node with the new value will be set at $I_{\min}$, which would give a second opportunity for broadcast relatively soon.

Note that the bottleneck scenario can be resolved within the Trickle algorithm by setting the redundancy constant on node 3 to a higher value ($k = 3$). However, in large networks, it is difficult to identify all possible bottlenecks and to manually configure them. Another solution is to use the previously described *adaptive-k* Trickle extension. However, even then, α should be carefully selected.

5.5.4 Evaluation

To confirm the analytical results and to evaluate the performance of the Cleansing MAC modifications, we perform several experiments in simulation and on a physical test bed. We use one application - dissemination of an update using Trickle, implemented in Contiki 2.7. Each experiment starts by injecting an update in the network. As the update is propagated, nodes increase their Trickle interval. The experiment ends when all nodes have reached their maximum Trickle interval $I_{\max} = 256s$. We

measure the delay, i.e. the time required to update all nodes, the total number of sent packets, the number of MAC layer retransmissions, and the mean waiting time in the MAC layer queue.

5.5.4.1 Simulation results

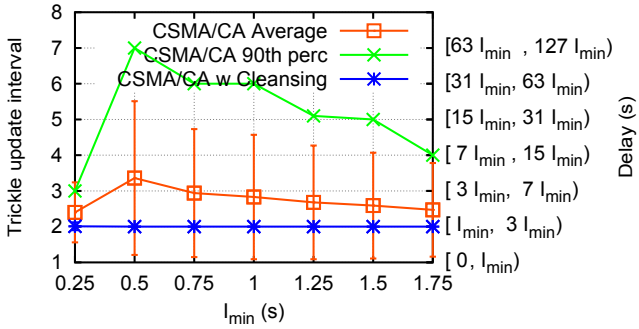
The simulations are carried out in the cross-level simulator Cooja. We use the UDGM propagation model, with no loss. All nodes use unslotted CSMA/CA with the default parameters ($BE_{\min} = 0$, $BE_{\max} = 3$, $NB_{\max} = 3$), and the ContikiMAC RDC protocol, with a wake-up frequency of 8Hz ($w = 125$ms). $I_{\min}$ varies from 250ms to 1.75s, at 250ms steps ($m = 2, 4, ..., 14$), well beyond MPL's recommendation of $m = 10$.

5.5.4.2 Bottleneck topology

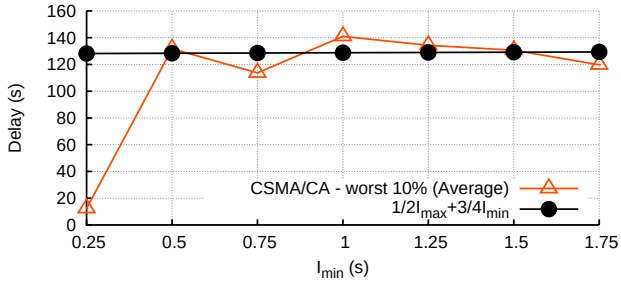
The first scenario follows the bottleneck topology, as shown in Figure 5.16. An update is inserted at the same time at nodes 1 and 2, and is propagated to the rest of the network using Trickle. Each configuration is simulated 1.000 times.

As expected, without Cleansing, due to the large number of collisions, the update delay of node 4 is highly variable (Figure 5.19a). Both the mean and the standard deviation peak at $I_{\min} = 0.5s$, and gradually decrease as $I_{\min}$ increases. Surprisingly, the update delay at $I_{\min} = 0.25s$ is stable. This anomaly occurs because at $I_{\min} = 0.25s = 2 \cdot w$, the contention window of nodes 1 and 2 is equal to the broadcast duration (w). This practically guarantees collisions, and a retransmission from one of the nodes. However, node 3's listen-only period will be finished before the retransmission starts, and there is a chance that node 3 will schedule its own transmission before it receives the retransmission. Even if the transmission from node 3 is delayed, it will be sent within one or two broadcast periods. However, with $I_{\min} = 0.5s$, node 1's and 2's contention window is still small, giving high probability for collisions. Then, retransmissions will always fall in node 3's listen-only period, forcing it to suppress its own transmission.

Figure 5.19b depicts the average measured delay of the worst 10% of the observations. This is a clear indication that harmful back-offs due to CSMA/CA are not uncommon, and that their effects can be detrimental



(a) Trickle update interval and update delay



(b) Average update delay - worst 10%

Figure 5.19: Update delay in the bottleneck scenario ($I_{\max} = 256s$, $k = 1$, $\eta = 1/2$). a) shows the Trickle interval in which node 4 gets updated, with and without Cleansing MAC improvements. The left y axis shows the Trickle doubling interval, and the right y axis the actual time. b) shows the average delay of the largest 10% of the measurements, and the analytical expected delay. The error bars correspond to the standard deviation.

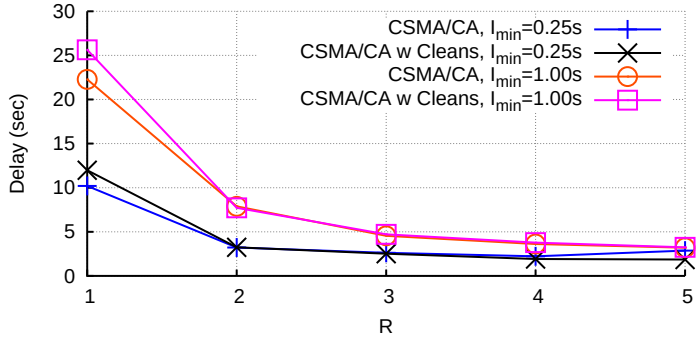
to Trickle's performance. The update delay then becomes significantly high, in line with the analytical expected delay of $\frac{3}{4}I_{\min} + \frac{1}{2}I_{\max}$.

Finally, the interference is completely resolved with MAC Cleansing. In that case, updates are always completed in the second interval, as expected.

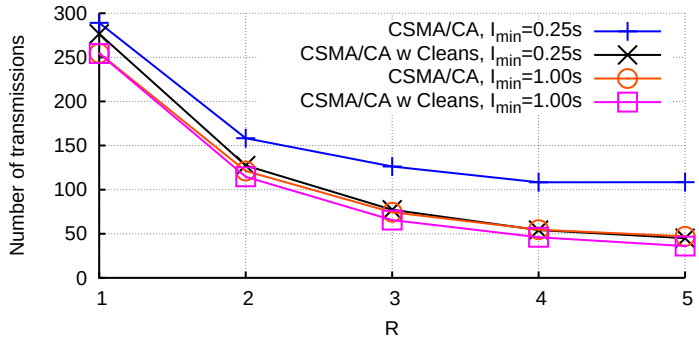
5.5.4.3 Grid topology

The second scenario consists of 100 nodes, arranged in a 10x10 grid, with 10 meters between two nodes in each axis. A new Trickle event is generated at the top left node. We simulate 100 executions of Trickle

with different values for $I_{\min}$. Furthermore, we varied the connectivity range of each node. Each node has a circular coverage area with radius $2 + 10R$ meters, with $1 \leq R \leq 5$.



(a) Update delay

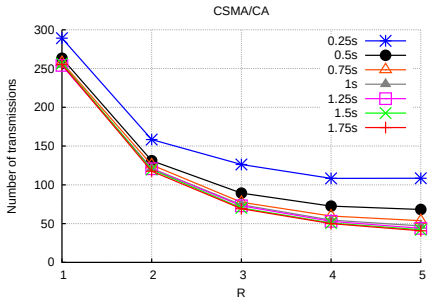


(b) Number of transmissions

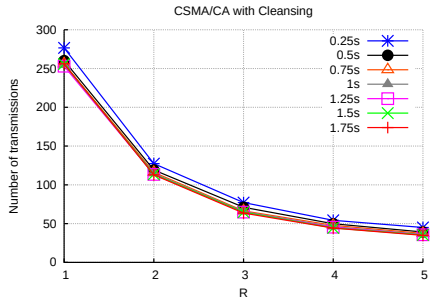
Figure 5.20: Average delay and average number of transmissions in the grid scenario. Using CSMA/CA with Cleansing with $I_{\min} = 0.25s$ requires a similar number of transmissions as regular CSMA/CA with $I_{\min} = 1.00s$, while the update delay is halved.

Figure 5.20a shows the update delay when using CSMA/CA with and without Cleansing. Since there are no bottlenecks in this scenario, these are comparable. However, the reduction in the number of sent packets is visible in Figure 5.20b. We can see that the number of transmissions with Cleansing is significantly lower than without Cleansing, while the average update delays are the same.

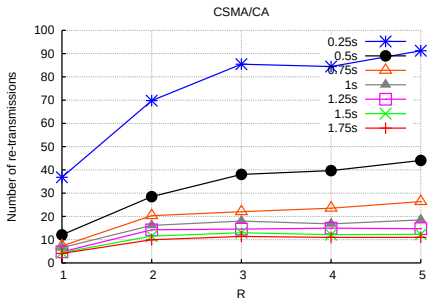
Figure 5.21 shows the average number of transmissions and retransmissions during the entire simulation. As the range of each node grows, fewer messages are required to cover the entire network. Trickle then



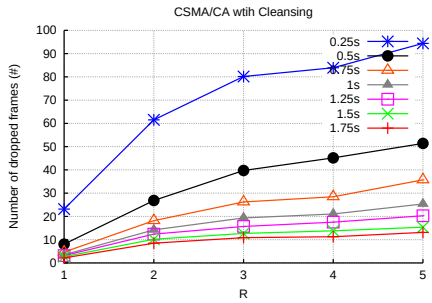
(a) TX - CSMA/CA



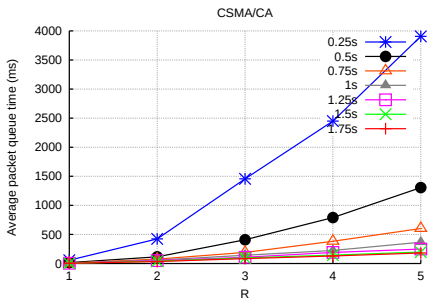
(b) TX - CSMA/CA w Cleans.



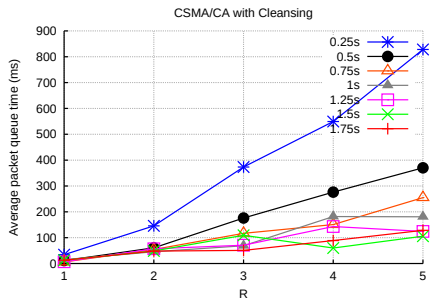
(c) RTX - CSMA/CA



(d) Dropped frames - CSMA/CA w Cleans.



(e) Frame queue time - CSMA/CA



(f) Frame queue time - CSMA/CA w Cleans.

Figure 5.21: Average number of transmissions, retransmissions and average frame queue time in the grid scenario, with (d, e, f) and without (a, b, c) MAC Cleansing, for different values of $I_{\min}$, $k = 1$ and $\eta = 1/2$.

performs well, suppressing many transmissions (Figure 5.21a). However, many of the messages are actual retransmissions from the MAC layer (Figure 5.21c). Since $k = 1$, these are obsolete messages. Furthermore, due to the congested media, frames are left in the queue for a longer time (Figure 5.21e), often leading to chained attempts for retransmission and further back-offs.

Figures 5.21b, 5.21d and 5.21f show the impact of using Cleansing. CSMA/CA with Cleansing is aggressive with cleaning the MAC queue, as is visible in Figure 5.21d. This makes Trickle work as intended even for small values of $I_{\min}$. Additionally, the average queue time is considerably lower compared to the original CSMA/CA.

5.5.4.4 Hardware experiments

To confirm the simulation results, we run the same application on the Grenoble physical test bed provided by FIT IoT-LAB. The experiment consists of 119 STM32 (ARM Cortex M3) based nodes, with the AT86RF231 IEEE 802.15.4 radio chip, arranged as in Figure 5.22. As before, all nodes use the ContikiMAC RDC protocol with a wake-up frequency of 8 Hz. The redundancy constant is fixed to $k = 1$, with $I_{\min}$ set to 0.25s, 0.5s and 1.0s. For each setting, we run 100 executions of Trickle dissemination of one update, injected at the bottom-right node.

Figure 5.23c shows that using CSMA/CA, low values of $I_{\min}$ introduce a lot of collisions, which force retransmissions by the MAC layer. Increasing $I_{\min}$ helps reduce the number of transmissions (Figure 5.23b), but at the expense of a higher delay (Figure 5.23a). On the other hand, CSMA/CA with Cleansing has consistent performance using all three different values of $I_{\min}$. Due to the proactive purging policy, the number of messages remains comparable with different values of $I_{\min}$. As expected, the delay increases together with $I_{\min}$, but it is still in the same range as with the original CSMA/CA.

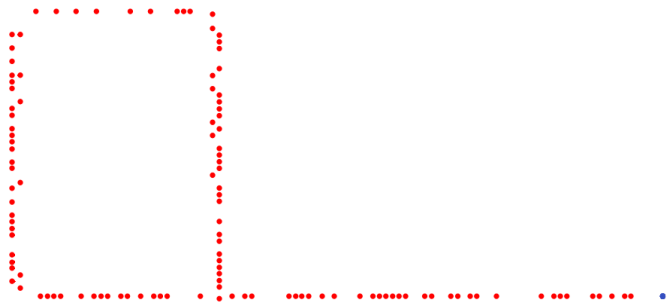
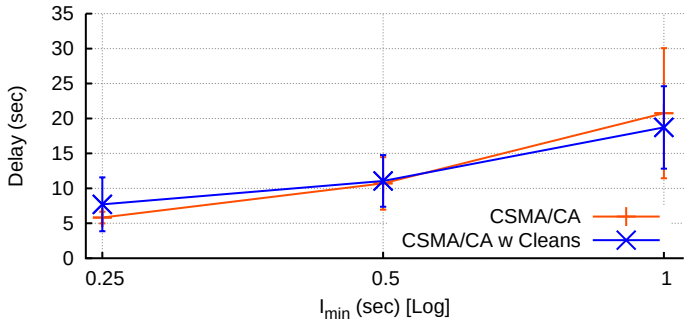
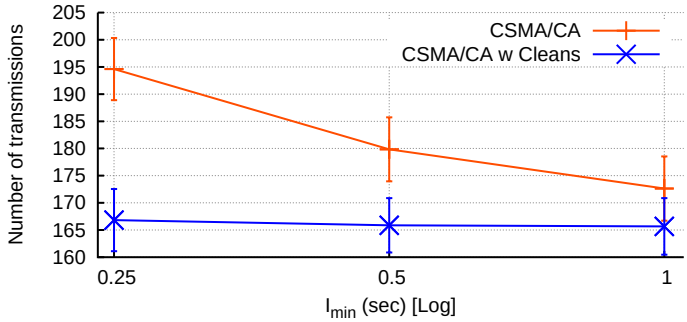


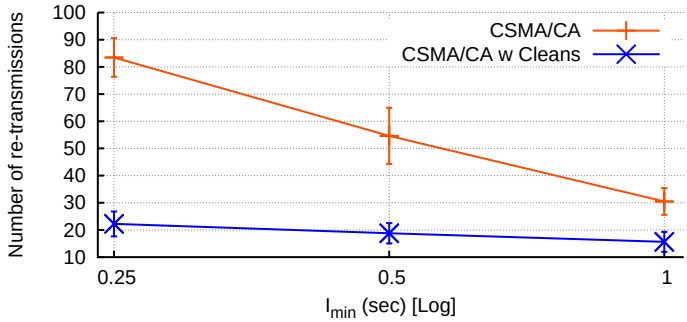
Figure 5.22: Topology of the IoT-Lab test bed. Experimental results from the IoT-Lab test bed. An update is injected at the bottom-right node, and is propagated using Trickle. The entire network is reachable in 12 hops.



(a) Delay



(b) Transmissions



(c) Retransmissions

Figure 5.23: Experimental results from the IoT-Lab test bed. We show the averages and standard deviations over 100 executions.

5.5.5 Summary

Even though Trickle was designed as a robust protocol, it still prone to errors in particular environments. As demonstrated, the performance of the Trickle algorithm for data dissemination can seriously suffer due to mis-communication with the link layer. The consequences vary, from increased number of transmissions to large update delays.

The proposed *Cleansing* modification is a lightweight method for limiting the influence of the link layer. *Cleansing* requires cross-layer operation, which can be feasible in resource constrained devices, where performance is critical.

5.6 Conclusion

In this chapter, we addressed research questions **RQ4** and **RQ5**, by analyzing the performance of the Trickle algorithm when used in routing (RPL) and multicast forwarding (MPL) protocols. These two protocols are of particular importance in the IoT and the system architecture proposed in Chapter 2, since they provide the basic communication primitives in LLNs. We showed that even though the Trickle algorithm is well studied, it is frequently observed to perform less than ideally in practice. We demonstrated two such problems.

The first problem arises from the static nature of the redundancy constant. In networks of varying densities, the Trickle algorithm favours nodes with fewer neighbours, which can lead to an unbalanced view of the network. To resolve the issue, we propose an extension to the algorithm, called *adaptive-k*, which allows nodes to set their own redundancy constant according to local information on network density. By simulations and experiments on a physical test bed, we showed that the *adaptive-k* extension improves the performance of the RPL routing protocol, by keeping the overhead of control traffic low, while still discovering good routes.

The second problem comes from the behaviour of the link layer. Due to the lack of communication between Trickle and the lower layers, delayed transmissions at the link layer can cause inconsistencies within the Trickle algorithm. These inconsistencies can result in increased overhead traffic, message loss, or starvation. RDC protocols further aggravates the

problem, virtually guaranteeing its occurrence in particular configurations. In order to resolve these issues, we proposed a small modification to the MAC layer, called *Cleansing*. The Cleansing MAC modification purges obsolete Trickle messages that are sent due to the inconsistencies caused by the MAC layer. Through a simulation study, and then confirmed with experiments on a large physical test bed, we showed that the Cleansing MAC indeed improves performance. We found that the number of redundant transmissions in dense topologies is decreased greatly and that the update speed in networks with bottlenecks is improved drastically.

6

CONCLUSION

This chapter concludes the thesis. We first give a summary of the problems explored in the thesis. Then, we highlight the research contributions which help solve the stated problems, and analyze their implications. Finally, we identify open problems for future work.

6.1 Contributions

The Internet of Things (IoT) changes the way how to think about our environment and how we perceive it. The IoT is built of physical objects embedded with electronics, sensors, actuators, software and connectivity modules. It is able to use such devices for communication among themselves, with other devices and with end-users, in order to build advanced applications. Even though these *smart* objects are of constrained nature, with limited processing, storage and communication capabilities, they can be programmed to react to the environment and jointly work towards reaching a common goal.

At this moment, we are at the beginning of the development of the IoT - we can see the first IoT implementations appearing: smart thermostats, connected cars with built-in sensors, smart cities, systems for infrastructure monitoring, and other similar systems. By 2020, it is expected that the number of connected devices in the IoT will reach 26 billion [Inc14], and this number will only continue to grow. Furthermore, the IoT has many potential applications, which combined with the heterogeneity of software and hardware components involved, the large number of devices and the amount of available data for processing, pose an immense problem. Therefore, in this thesis we tried to answer several questions about the system architecture of IoT systems, as follows.

RQ1 How do we design the software infrastructure for the IoT?

We focused on the design of the system architecture in **Chapter 2**. We identified the key constraints which have to be met by potential software protocols, applications and systems for the IoT, by looking at the various IoT use cases, application design styles and the heterogeneous devices which comprise it. The main points from this survey are that potential solutions have to scale to large networks, be portable to constrained devices, and be open to various vendors, users and domains. Therefore, we believe that the definition and adoption of open standards across all software layers is of paramount importance for the success of the IoT.

RQ2 How do we combine powerful and resource constrained devices in a single logical network?

We partially addressed this question in chapters 2 and 4. In **Chapter 2**, based on the given constraints, we proposed adopting a Service Oriented Architecture (SOA) for building IoT systems. In this architecture,

applications are built by connecting different *services*. The fundamental functionality, as connectivity, data delivery, service discovery and management, needed to build both applications and services, is provided by a standardized *platform*. Both services and applications can be added and removed in the network at runtime. The benefit of this architecture is that it is flexible enough to scale with the size and the distinct application design styles of the IoT. Furthermore, it can support both resource constrained and powerful devices in the same network, by distributing workload based on the resources available at individual devices. To illustrate its applicability, we described a potential implementation of the system architecture for a building automation system, built on top of a low-power wireless network using standards based protocols. From the implementation, we identified three important issues which are further analyzed in the following chapters: software updates, service discovery and data dissemination.

In **Chapter 4**, we focused on software protocols for service discovery in the IoT. Service discovery is a crucial feature of the proposed architecture, since it allows devices to automatically discover available services in a network, and learn by themselves how to access them. We identified that service discovery protocols have to balance between complexity and available features. Even though many such protocols have been developed so far, we found that none of them are ideal: they are either too specific for resource constrained devices, they have not been widely accepted, or they are too resource demanding. To resolve this issue, we proposed using the Multicast Domain Name System (mDNS) with DNS-Based Service Discovery (DNS-SD) protocol, with two extensions for improved operation. The first extension of the protocol enables the inclusion of low-power devices using proxy servers, which addresses research question **RQ2**. The second extension improves the scalability of the protocol, and addresses research question **RQ4**.

RQ3 How do we manage large networks of heterogeneous devices?

We tackled this question in **Chapter 3**, by analyzing the problem of updating software in large low-power networks. This has long been seen as a tedious task due to the size of the software updates, the lossy nature of the links in the network, and the limited energy available on devices. We showed that by applying different data compression techniques, notably incremental updates, the reduction in the size of the software updates for delivery far overweighs the required processing

effort involved, both in time and in energy usage. We demonstrated that further reductions in the size of software updates can be achieved by preparing and distributing updates of similar devices together. The involved algorithms are simple enough to be implemented even on the most constrained devices, which supports the decisions made in Chapter 2, and enables the creation of an adaptable system architecture.

RQ4 How do existing IoT specific networking protocols scale in size and in different topologies?

We addressed this question for two protocols in Chapters 4 and 5. In **Chapter 4**, we showed that the mDNS/DNS-SD protocol for service discovery contains only simple rules for selecting services. As a result, in large networks with many similar service providers, the protocol generates a lot of overhead traffic for service discovery. To resolve this issue, we proposed a second extension of the protocol. This extension improves the descriptiveness of the protocol, by allowing services to be discovered based on their context. As a result, this extensions improves the scalability of the protocol in large networks with many services.

In **Chapter 5**, we analyzed two existing protocols under standardization for routing and multicast forwarding in low-power networks: IPv6 Routing Protocol for LLNs (RPL) and Multicast Protocol for LLNs (MPL). Both protocols are based on the Trickle algorithm for disseminating traffic in ad-hoc networks with little overhead. We showed that due to the design of the Trickle algorithm, the protocols can have sub-optimal performance in various network topologies. In the first case, we showed that the Trickle algorithm favours nodes with few neighbours to transmit more frequently. This unfairness can lead to suboptimal routing in dense networks, by forcing routes over nodes with fewer neighbours. We resolved this problem by modifying the Trickle algorithm. In the second case, multicast forwarding can suffer due to the lack of communication between the network and the link layer. We resolved this interference by modifying the link layer. The second case answers the last research question **RQ5**.

RQ5 What is the impact of low-power wireless radios on different IoT applications?

All contributions presented in this thesis improve the state-of-the art in protocol design for the IoT. The described methods and algorithms improve performance of existing algorithms based on the fact that limited

computation is preferred to excessive networking traffic in large low-power networks. This is important design criteria, considering that the IoT will span to billions of devices, and radio traffic is the main source of energy drainage. The presented results help with the implementation of the IoT, by enhancing its scalability to large networks.

6.2 Limitations and future work

The broadness of the IoT gives a lot of questions that should be addressed. Even though we tackled many of them with this work, we still have many open problems. Furthermore, our work generated some new interesting areas for exploration. We now present some of these questions, sorted by the chapter in which they first appear.

System architecture. The first IoT implementations provide valuable feedback for the design of future systems. Based on this feedback, we believe that the next step will be standardization of common features required by all IoT systems, and using these features as integration points for heterogeneous systems. We listed several such common features in the design of the *platform* in Chapter 2, but this list can be further extended, with different IoT applications in mind. For example, the current platform lacks means for validating and enforcing functional requirements, such as deadlines and resource demands. These requirements are of importance for critical applications, including safety and security systems.

Software update. In Chapter 3, all data compression and incremental update algorithms were taken as black boxes. Therefore, additional compression can be achieved if they are customized for the target code and platform. For instance, the compiler can be adapted to generate firmware images which are optimized for incremental updates, as described in [PBM11].

Similarly, the presented method for horizontal patching was done agnostic to the networking topology, and only the size of the updates was taken as a metric. However, if the networking topology is known, then the combined horizontal deltas can be tailored to fit distribution paths. The main benefit of this approach is that at leaf nodes, only the single vertical deltas would be distributed, instead of the entire combined horizontal deltas.

Service discovery. We believe that the future of the mDNS/DNS-SD protocol for service discovery in the IoT, lies in a binary transport format, much like HTTP 2.0. The human-readable nature of the protocol can be provided by APIs, while the transport itself can be in an optimized binary format. The compression method presented in Chapter 4 is a possible way to achieve this goal, but it requires further evaluation.

A limitation of the analysis of the different proxy registration protocols is that it was done only in a single-hop network. Therefore, it should be further extended for multi-hop networks, where the multicast forwarding protocol, the routing protocol, and the network topology have an important impact on performance. Such analytical extensions are also interesting for the design of cross-layer solutions, as the integration of service discovery and multicast grouping. For instance, in Chapter 2 we described that the logical grouping between different devices is done using the service discovery protocol, based on the context of the devices involved. For better performance, this binding at the service discovery layer can be translated into implicit grouping at the networking layer, by generating multicast group addresses based on context. Then, the scope of service discovery queries would hopefully be confined to a much smaller region, improving performance.

Trickle-based protocols. In the evaluation of the Trickle algorithm in Chapter 5, we identified a key problem in the lack of communication between protocols at different software layers. We presented a specific solution to the described problem, but we believe that it should be generalized. In particular, this generalization should be in two directions: a generalized interference model and a generalized communication model. The former model should cover interference scenarios between arbitrary protocols running at different software layers, such as a Medium Access Control (MAC) protocol on top of a Radio Duty Cycling (RDC) protocol. As a result, it should provide probabilities for interference scenarios to occur, and their impact. The latter model concerns the design of primitives required between different layers. As we showed, the current set of primitives provided by the link layer are insufficient for proper operation of the networking layer. Furthermore, various implementations expose distinct interfaces, hindering interoperability. Therefore, for improved operation, these primitives should be first extended, and then standardized.

A limitation of our analysis of multicast forwarding is that we only considered a single event being propagated, with the redundancy constant

fixed to one. In practice, multiple events are expected, which would generate multiple flows, possibly of different origin. Therefore, another option for future work is to evaluate the protocol in such scenarios, taking into consideration the phasing of concurrent forwarding waves, and the time required for the network to stabilize.

With the introduction of the adaptive redundancy constant, we simplified the configuration of the Trickle algorithm. The next step is the automatic adaptation of other Trickle parameters, such as the minimum Trickle interval. We already have some guidelines for setting it, as it should be correlated to the actual value of the redundancy constant, for which we presented analytical results. If successful, this would lead to the creation of a virtually configure-less algorithm, which is of huge importance for the IoT, where it is already seen as a new communication primitive.

BIBLIOGRAPHY

[AGBC11] Nicola Accettura, Luigi Alfredo Grieco, Gennaro Boggia, and Pietro Camarda. "Performance analysis of the RPL Routing Protocol". In: *IEEE Conference on Mechatronics*. ICM. 2011, pp. 767–772. DOI: 10.1109/ICMECH.2011.5971218.

[Agg14] Ashutosh Aggarwal. *Optimizing DNS-SD query using TXT records*. http://tools.ietf.org/html/draft-aggarwal-dnssd-optimize-query. Internet Engineering Task Force, 2014.

[AH14] Mohammad Aazam and Eui-Nam Huh. "Fog Computing and Smart Gateway Based Communication for Cloud of Things". In: *Conference on Future Internet of Things and Cloud*. FICLOUD '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 464–470. ISBN: 978-1-4799-4357-9. DOI: 10.1109/FiCloud.2014.83.

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. "The Internet of Things: A survey". In: *Computer Networks* 54.15 (2010), pp. 2787 –2805. ISSN: 1389-1286. DOI: http://dx.doi.org/10.1016/j.comnet.2010.05.010.

[AIV10] Diana Albu, Johan J. Lukkien, and Richard Verhoeven. "Energy effect of on-node processing of ECG signals". In: *Conference on Consumer Electronics*. ICCE. 2010, pp. 7–8. DOI: 10.1109/ICCE.2010.5418748.

[Anc+14] Emilio Ancillotti et al. "Trickle-L2: Lightweight link quality estimation through Trickle in RPL networks". In: *15th IEEE Symposium on A World of Wireless, Mobile and Multimedia Networks*. WoWMoM. 2014, pp. 1–9. DOI: 10.1109/WoWMoM.2014.6918951.

[ARYK10] Fatima Muhammad Anwar, Muhammad Taqi Raza, Seung-Wha Yoo, and Ki-Hyung Kim. "ENUM Based Service Discovery Architecture for 6LoWPAN". In: *Wireless Communications and Networking Conference*. WCNC. 2010, pp. 1 –6. DOI: 10.1109/WCNC.2010.5506568.

[ASGT12] Vlado Altmann, Jan Skodzik, Frank Golatowski, and Dirk Timmermann. "Investigation of the use of embedded Web Services in smart metering applications". In: *Conference on IEEE Industrial Electronics Society*. IECON. Oct. 2012, pp. 6172–6177. DOI: 10.1109/IECON.2012.6389071.

[Ash09] Kevin Ashton. *That 'Internet of Things' Thing*. [Online] http://www.rfidjournal.com/articles/view?4986. June 2009.

[Baz+12] Marco Bazzani et al. “Enabling the IoT Paradigm in E-health Solutions through the VIRTUS Middleware”. In: *Conference on Trust, Security and Privacy in Computing and Communications*. TrustCom. June 2012, pp. 1954–1959. DOI: 10.1109/TrustCom.2012.144.

[BBCF06] John Buford, Bernard Burg, Emre Celebi, and Phyllis G. Frankl. “Sleeper: A Power-Conserving Service Discovery Protocol”. In: *Conference on Mobile and Ubiquitous Systems*. 2006, pp. 1–9. DOI: 10.1109/MOBIQW.2006.361725.

[Ben+13] Sven Bendel et al. “A service infrastructure for the Internet of Things based on XMPP”. In: *Int. Conference on Pervasive Computing and Communications Workshops*. PERCOM Workshops. Mar. 2013, pp. 385–388. DOI: 10.1109/PerComW.2013.6529522.

[BKG11] Markus Becker, Koojana Kuladinithi, and Carmelita Görg. “Modelling and Simulating the Trickle Algorithm”. In: *Conference on Mobile Networks and Management*. MONAMI. 2011, pp. 135–144.

[BL12] Michael Blackstock and Rodger Lea. “IoT mashups with the WoTKit”. In: *Conference on Internet of Things*. IOT. Oct. 2012, pp. 159–166. DOI: 10.1109/IOT.2012.6402318.

[Blu12] Bluetooth special interest group. *Core version 4.0*. 2012.

[BLV09] Remi Bosman, Johan J. Lukkien, and Richard Verhoeven. “An Integral Approach to Programming Sensor Networks”. In: *Consumer Communications and Networking Conference*. CCNC. 2009, pp. 1–5. DOI: 10.1109/CCNC.2009.4784846.

[BLV11] Remy Bosman, Johan J. Lukkien, and Richard Verhoeven. “Gateway architectures for service oriented application-level gateways”. In: *IEEE Tran. on Consumer Electronics,* 57.2 (2011), pp. 453–461. DOI: 10.1109/TCE.2011.5955179.

[BMNZ14] Flavio Bonomi, Rodolfo Milito, Preethi Natarajan, and Jiang Zhu. “Fog Computing: A Platform for Internet of Things and Analytics”. English. In: *Big Data and Internet of Things: A Roadmap for Smart Environments*. Ed. by Nik Bessis and Ciprian Dobre. Vol. 546. Studies in Computational Intelligence. Springer International Publishing, 2014, pp. 169–186. ISBN: 978-3-319-05028-7. DOI: 10.1007/978-3-319-05029-4_7.

[BMT03] Paolo Bellavista, Rebecca Montanari, and Daniela Tibaldi. “COSMOS: A Context-Centric Access Control Middleware for Mobile Environments”. English. In: *Mobile Agents for Telecommunication Applications*. Ed. by Eric Horlait, Thomas Magedanz, and RochH. Glitho. Vol. 2881. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2003, pp. 77–88. ISBN: 978-3-540-20298-1. DOI: 10.1007/978-3-540-39646-8_8.

[Bot11] Andre Bottaro. “Open The Box! Challenges in the Smart Home (Keynote speech)”. In: *IEEE Emerging Technologies and Factory Automation*. ETFA. 2011.

[BPGO12] Talal Ashraf Butt, Iain Phillips, Lin Guan, and George Oikonomou. “TRENDY: An Adaptive and Context-Aware Service Discovery Protocol for 6LoWPANs”. In: *Workshop Web of Things*. WoT. 2012.

[Bra+08] Tim Bray et al. *W3C Recommendation: Extensible Markup Language (XML)*. http://www.w3.org/TR/xml/. World Wide Web Consorcium, 208.

[BRFB11] Tegawendé F. Bissyandé, Laurent Réveillère, Jean-Rémy Falleri, and Yérom-David Bromberg. “Typhoon: A Middleware for Epidemic Propagation of Software Updates”. In: *Workshop Middleware for Pervasive Mobile and Embedded Computing*. M-MPAC. Lisbon, Portugal: ACM, 2011, 1:1–1:7. ISBN: 978-1-4503-1065-9. DOI: 10.1145/2090316.2090317.

[Bru+11] AJ Brush et al. “Home automation in the wild: challenges and opportunities”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2011, pp. 2115–2124.

[Buc14] Nina Buchina. *Extending service discovery protocols with support for context information*. Master Thesis, Eindhoven University of Technology. 2014.

[But14] Talal A. Butt. “Provision of adaptive and context-aware service discovery for the Internet of Things”. PhD thesis. Loughborough University, 2014.

[BYAH06] Michael Buettner, Gary V. Yee, Eric Anderson, and Richard Han. “X-MAC: A Short Preamble MAC Protocol for Duty-cycled Wireless Sensor Networks”. In: *Conference on Embedded Networked Sensor Systems*. SenSys. 2006, pp. 307–320. ISBN: 1-59593-343-3. DOI: 10.1145/1182807.1182838.

[Cay89] Arthur Cayley. “A Theorem on Trees”. In: *Quarterly Jounral of Pure and Applied Mathematics* 23 (1889), pp. 376 –378.

[Cha+08] Sajjad Hussain Chauhdary et al. “A Context-Aware Service Discovery Consideration in 6LoWPAN”. In: *Conference on Convergence and Hybrid Information Technology*. IEEE Computer Society, 2008, pp. 21–26. ISBN: 978-0-7695-3407-7.

[CJAK06] Shafique Ahmad Chaudhry, Won Do Jung, Ali Hammad Akbar, and Ki-Hyung Kim. “Proxy-Based Service Discovery and Network Selection in 6LoWPAN”. In: *High Performance Computing and Communications*. Vol. 4208. Lecture Notes in Computer Science. 2006, pp. 525–534. ISBN: 978-3-540-39368-9. DOI: 10.1007/11847366_54.

[CK06] Stuart Cheshire and Marc Krochmal. *Dynamic DNS Update Leases*. Internet Engineering Task Force, 2006.

[CK09] Stuart Cheshire and Marc Krochmal. *EDNS0 OWNER Option*. Internet Engineering Task Force, 2009.

[Cor13] Echelon Corporation. *Industrial Internet of Things*. http://www.digikey.nl/en/pdf/e/echelon/iiot-wp. [Online; accessed 13-April-2015]. 2013.

[CT03] Inc Crossbow Technology. *Mote In-Network Programming User Reference Version 20030315*. 2003.

[CVY13] Thomas Clausen, Axel Colin de Verdiere, and Jiazi Yi. "Performance Analysis of Trickle as a Flooding Mechanism". In: *Conference on Communication Technology*. ICCT. 2013. DOI: 10.1109/ICCT.2013.6820439.

[Dan+15] Conrad Dandelski et al. "Scalability of dense wireless lighting control networks". In: *IEEE Communications Magazine* 53.1 (2015), pp. 157–165. DOI: 10.1109/MCOM.2015.7010529.

[DB10] Kirsten Dolfus and Torsten Braun. "An evaluation of compression schemes for wireless networks". In: *Congress on Ultra Modern Telecommunications and Control Systems and Workshops*. ICUMT. 2010, pp. 1183–1188. DOI: 10.1109/ICUMT.2010.5676532.

[Deu+06] Scott de Deugd et al. "SODA: Service Oriented Device Architecture". In: *IEEE Pervasive Computing* 5.3 (July 2006), pp. 94–96. ISSN: 1536-1268. DOI: 10.1109/MPRV.2006.59.

[DGV04] Adam Dunkels, Björn Grönvall, and Thiemo Voigt. "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors". In: *Workshop on Embedded Networked Sensors*. Emnets-I. 2004. DOI: 10.1109/LCN.2004.38.

[DL03] Tijs van Dam and Koen Langendoen. "An Adaptive Energy-efficient MAC Protocol for Wireless Sensor Networks". In: *Conference on Embedded Networked Sensor Systems*. SenSys. 2003, pp. 171–180. ISBN: 1-58113-707-9. DOI: 10.1145/958491.958512.

[Dun11a] Adam Dunkels. *Powertrace: Network-level Power Profiling for Low-power Wireless Networks*. Tech. rep. SICS T2011:15, 2011.

[Dun11b] Adam Dunkels. *The ContikiMAC Radio Duty Cycling Protocol*. Tech. rep. SICS T2011:13, 2011.

[Dut+10] Prabal Dutta et al. "Design and Evaluation of a Versatile and Efficient Receiver-initiated Link Layer for Low-power Wireless". In: *Conference on Embedded Networked Sensor Systems*. SenSys. Zurich, Switzerland, 2010, pp. 1–14. ISBN: 978-1-4503-0344-6. DOI: 10.1145/1869983.1869985.

[EG14] Joakim Eriksson and Omprakash Gnawali. “Poster Abstract: Synchronizing Trickle Intervals”. In: *European Conference on Wireless Sensor Networks*. EWSN. 2014.

[EHD04] Amre El-Hoiydi and Jean-Dominique Decotignie. “WiseMAC: an ultra low power MAC protocol for the downlink of infrastructure wireless sensor networks”. In: *Symposium on Computers and Communications*. Vol. 1. ISCC. 2004, 244–251 Vol.1. DOI: 10.1109/ISCC.2004.1358412.

[EJS13] Atis Elsts, Janis Judvaitis, and Leo Selavo. “SEAL: A Domain-Specific Language for Novice Wireless Sensor Network Programmers”. In: *EUROMICRO Conference on Software Engineering and Advanced Applications*. SEAA. Sept. 2013, pp. 220–227. DOI: 10.1109/SEAA.2013.16.

[Eur11] European Telecommunications Standards Institute. *ETSI TS 102 690 V1.1.1 Machine-to-Machine communications (M2M): Functional architecture*. [Online] http://www.etsi.org/. 2011.

[Eur14] European Telecommunications Standards Institute. *ETSI TS 102 921 V1.3.1 Machine-to-Machine communications (M2M); mIa, dIa and mId interfaces*. [Online] http://www.etsi.org/. 2014.

[FGHHV01] Hendrik C. Ferreira, Henricus M. Grové, Olaf Hooijen, and A. J. Han Vinck. *Power Line Communication*. John Wiley & Sons, Inc., 2001. ISBN: 9780471346081. DOI: 10.1002/047134608X.W2004.

[Fin03] Klaus Finkenzeller. *RFID Handbook: Fundamentals and Applications in Contactless Smart Cards and Identification*. 2nd ed. New York, NY, USA: John Wiley & Sons, Inc., 2003. ISBN: 0470844027.

[FKOJ14] Stefan Forsstrom, Victor Kardeby, Patrik Osterberg, and Ulf Jennehag. “Challenges when Realizing a Fully Distributed Internet-of-Things - How we Created the SensibleThings Platform”. In: *Conference on Digital Telecommunications*. ICDT. IARIA, 2014, pp. 13–18. ISBN: 978-1-61208-316-2.

[GBMP13] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. “Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions”. In: *Future Generation Computer Systems* 29.7 (Sept. 2013), pp. 1645–1660. ISSN: 0167-739X. DOI: 10.1016/j.future.2013.01.010.

[Ger14] Jack Germain. *The Importance of Openness to the Internet of Things*. Ed. by Linuxinsider.com. [Online; posted 09-November-2014] http://www.linuxinsider.com/story/81024.html. Nov. 2014.

[GGBECF14] Cristian Gonzalez Garcia, B. Cristina Pelayo G-Bustelo, Jordán Pascual Espada, and Guillermo Cueva-Fernandez. "Midgar: Generation of heterogeneous objects interconnecting applications. A Domain Specific Language proposal for Internet of Things scenarios ". In: *Computer Networks* 64.0 (2014), pp. 143 – 158. ISSN: 1389-1286. DOI: http://dx.doi.org/10.1016/j.comnet.2014.02.010.

[Gna+09] Omprakash Gnawali et al. "Collection Tree Protocol". In: *Conference on Embedded Networked Sensor Systems*. SenSys. Berkeley, CA, USA, 2009.

[Gre14] John Greenough. *The Enterprise Internet of Things Report: Forecasts, Industry Trends, Advantages, and Barriers for the Top IoT Sector*. Business Insider. [Online] https://intelligence.businessinsider.com/the-enterprise-internet-of-things-report-forecasts-industrytrends-advantages-and-barriers-for-the-top-iot-sector-2014-11. 2014.

[GT09] Dominique Guinard and Vlad Trifa. "Towards the Web of Things: Web Mashups for Embedded Devices". In: *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences)*. Madrid, Spain, Apr. 2009.

[HC02] Jason L. Hill and David E. Culler. "Mica: A Wireless Platform for Deeply Embedded Networks". In: *IEEE Micro* 22.6 (Nov. 2002), pp. 12–24. ISSN: 0272-1732. DOI: 10.1109/MM.2002.1134340.

[HC08] Jonathan W. Hui and David E. Culler. "IP is Dead, Long Live IP for Wireless Sensor Networks". In: *Conference on Embedded Network Sensor Systems*. SenSys. Raleigh, NC, USA: ACM, 2008, pp. 15–28. ISBN: 978-1-59593-990-6. DOI: 10.1145/1460412.1460415.

[HDVL03] Sumi Helal, Nitin Desai, Varun Verma, and Choonhwa Lee. "Konark - a service discovery and delivery protocol for ad-hoc networks". In: *Wireless Communications and Networking*. Vol. 3. WCNC. 2003, pp. 2107–2113. DOI: 10.1109/WCNC.2003.1200712.

[HK14] Jonathan Hui and Richard Kelsey. *Multicast Protocol for Low power and Lossy Networks (MPL)*. http://tools.ietf.org/html/draft-ietf-roll-trickle-mcast-09. Internet Engineering Task Force, 2014.

[Hon+13] Kirak Hong et al. "Mobile Fog: A Programming Model for Large-scale Applications on the Internet of Things". In: *ACM SIGCOMM Workshop on Mobile Cloud Computing*. MCC. Hong Kong, China: ACM, 2013, pp. 15–20. ISBN: 978-1-4503-2180-8. DOI: 10.1145/2491266.2491270.

[IEEE1471] IEEE. *IEEE Recommended Practice for Architectural Description for Software-Intensive Systems*. 2000.

[IEEE15.4e] IEEE. *Standard for Local and metropolitan area networks — Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs), Amendment 1: MAC sublayer*. 2012.

[IEEE3bt] IEEE. *Standard for Ethernet Amendment: Physical Layer and Management Parameters for DTE Power via MDI over 4-Pair*. 2013.

[Inc14] Gartner Inc. *Gartner Says a Thirty-Fold Increase in Internet-Connected Physical Devices by 2020 Will Significantly Alter How the Supply Chain Operates*. Ed. by Gartner.com. [Online; posted 24-March-2014] `http://www.gartner.com/newsroom/id/2688717`. Mar. 2014.

[Inc15] LogMeIn Inc. *Xively*. http://www.xively.com. [Online; accessed 01-June-2015]. 2015.

[ITN13] Oana Iova, Fabrice Theoleyre, and Thomas Noel. "Stability and Efficiency of RPL under Realistic Conditions in Wireless Sensor Networks". In: *24th IEEE Symposium on Personal Indoor and Mobile Radio Communications*. PIMRC. 2013, pp. 2098–2102.

[JC04] Jaein Jeong and David Culler. "Incremental network programming for wireless sensors". In: *Conference on Sensor and Ad Hoc Communications and Networks*. SECON. 2004, pp. 2–33. DOI: `10.1109/SAHCN.2004.1381899`.

[Jen06] Jennic. *Application note JN-AN-1035: Calculating 802.15.4 Data Rates*. `http://www.jennic.com/files/support_files/JN-AN-1035%20Calculating%20802-15-4%20Data%20Rates-1v0.pdf`. 2006.

[JRK12] Markus Jung, Christian Reinisch, and Wolfgang Kastner. "Integrating Building Automation Systems and IPv6 in the Internet of Things". In: *Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. IMIS. July 2012, pp. 683–688. DOI: `10.1109/IMIS.2012.134`.

[JVVG14] Nildo Ribeiro Junior, Marcos A. M. Vieira, Luiz F. M. Vieira, and Omprakash Gnawali. "CodeDrip: Data Dissemination Protocol with Network Coding for Wireless Sensor Networks". In: *European Conference on Wireless Sensor Networks*. EWSN. 2014.

[KBBG11] Koojana Kuladinithi, Olaf Bergmann, Thomas Pötsch Markus Becker, and Carmelita Görg. "Implementation of CoAP and its Application in Transport Logistics". In: *Workshop on Extending the Internet to Low power and Lossy Networks*. IP+SN '11. Apr. 2011.

[KDD11] Matthias Kovatsch, Simon Duquennoy, and Adam Dunkels. “A Low-Power CoAP for Contiki”. In: *IEEE International Conference on Mobile Adhoc and Sensor Systems, MASS 2011, Valencia, Spain, October 17-22, 2011*. 2011, pp. 855–860. DOI: 10.1109/MASS.2011.100.

[KG14] Hamidreza Kermajani and Carles Gomez. “On the Network Convergence Process in RPL over IEEE 802.15.4 Multihop Networks: Improvement and Trade-Offs”. In: *Sensors* 14.7 (2014), pp. 11993–12022. ISSN: 1424-8220. DOI: 10.3390/s140711993.

[KGA12] Hamidreza R. Kermajani, Carles Gomez, and Mostafa Hesami Arshad. “Modeling the Message Count of the Trickle Algorithm in a Steady-State, Static Wireless Sensor Network”. In: *Communications Letters, IEEE* 16.12 (2012), pp. 1960–1963.

[KGKS11] Ronny Klauck, Jan Gaebler, Michael Kirsche, and Sebastian Schoepke. “Mobile XMPP and cloud service collaboration: An alliance for flexible disaster management”. In: *Conference on Collaborative Computing: Networking, Applications and Worksharing*. CollaborateCom. Oct. 2011, pp. 201–210.

[Kim+10] Ki Hyung Kim et al. *Simple Service Location Protocol (SSLP) for 6LoWPAN*. https://tools.ietf.org/html/draft-daniel-6lowpan-sslp. Internet Engineering Task Force, 2010.

[KK12a] Michael Kirsche and Ronny Klauck. “Unify to bridge gaps: Bringing XMPP into the Internet of Things”. In: *Conference on Pervasive Computing and Communications Workshops*. PERCOM Workshops. Mar. 2012, pp. 455–458. DOI: 10.1109/PerComW.2012.6197534.

[KK12b] Ronny Klauck and Michael Kirsche. “Bonjour Contiki: A Case Study of a DNS-based Discovery Service for the Internet of Things”. In: *Conference on Ad-hoc, Mobile, and Wireless Networks*. ADHOC-NOW. 2012, pp. 316–329. ISBN: 978-3-642-31637-1. DOI: 10.1007/978-3-642-31638-8_24.

[KK13] Ronny Klauck and Michael Kirsche. “Enhanced DNS Message Compression - Optimizing mDNS/DNS-SD for the Use in 6LoWPANs”. In: *Workshop on Sensor Networks and Systems for Pervasive Computing*. PerSeNS. 2013.

[KLD12] Matthias Kovatsch, Martin Lanter, and Simon Duquennoy. “Actinium: A restful runtime container for scriptable internet of things applications”. In: *Conference on the Internet of Things*. IOT. IEEE. 2012, pp. 135–142.

[KLKG15] Evgeny Khorov, Andrey Lyakhov, Alexander Krotov, and Andrey Guschin. “A survey on IEEE 802.11ah: An enabling networking technology for smart cities”. In: *Computer Communications* 58.0 (2015). Special Issue on Networking and Communications for Smart Cities, pp. 53 –69. ISSN: 0140-3664. DOI: http://dx.doi.org/10.1016/j.comcom.2014.08.008.

[KM10] Ryozo Kiyohara and Satoshi Mii. “BPE Acceleration Technique for S/W Update for Mobile Phones”. In: *Conference on Advanced Information Networking and Applications*. AINA. 2010, pp. 592–599. DOI: 10.1109/AINA.2010.65.

[Ko+11] Jeonggil Ko et al. “Evaluating the Performance of RPL and 6LoWPAN in TinyOS”. In: *Extending the Internet to Low power and Lossy Networks*. IPSN. 2011.

[KTT12] Ryozo Kiyohara, Koichi Tanaka, and Yoshiaki Terashima. “S/W upgrade for on-vehicle information devices”. In: *Conference on Consumer Electronics*. ICCE. 2012, pp. 19–20. DOI: 10.1109/ICCE.2012.6161718.

[L+12] Valerie Lampkin et al. *Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry*. IBM Redbooks, 2012.

[LC02] Philip Levis and David Culler. “Maté: a tiny virtual machine for sensor networks”. In: *Conference on Architectural support for programming languages and operating systems*. ASPLOS-X. 2002, pp. 85–95. ISBN: 1-58113-574-2. DOI: http://doi.acm.org/10.1145/605397.605407.

[Lev+05] Philip Levis et al. “TinyOS: An operating system for sensor networks”. In: *Ambient Intelligence*. Springer, 2005, pp. 115–148.

[Lev+08] Philip Levis et al. “The Emergence of a Networking Primitive in Wireless Sensor Networks”. In: *Communications of the ACM* 51.7 (2008), pp. 99–106. ISSN: 0001-0782.

[LG13] Philip Levis and Omprakash Gnawali. *Recommendations for Efficient Implementation of RPL*. https://tools.ietf.org/html/draft-gnawali-roll-rpl-recommendations. 2013.

[LL08] Kaisen Lin and Philip Levis. “Data Discovery and Dissemination with DIP”. In: *Conference on Information Processing in Sensor Networks*. IPSN. 2008, pp. 433–444. DOI: 10.1109/IPSN.2008.17.

[LPCS04] Philip Levis, Neil Patel, David Culler, and Scott Shenker. “Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks”. In: *Symposium on Networked Systems Design and Implementation*. NSDI. 2004, pp. 15–28.

[LVHD14] Chi-Anh La, Liviu-Octavian Varga, Martin Heusse, and Andrzej Duda. “Energy-efficient Multi-hop Broadcasting in Low Power and Lossy Networks”. In: *ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. MSWiM. Montreal, QC, Canada, 2014, pp. 41–50. ISBN: 978-1-4503-3030-5.

[Mac+06] C. Matthew MacKenzie et al. *Reference Model for Service Oriented Architecture 1.0*. OASIS. [Online] https://www.oasis-open.org/. 2006.

[MBBD14] Thomas M. M. Meyfroyt, Sem C. Borst, Onno J. Boxma, and Dee Denteneer. “Data Dissemination Performance in Large-scale Sensor Networks”. In: *SIGMETRICS Performance Evaluation Review* 42.1 (2014), pp. 395–406.

[MBBD15] Thomas M. M. Meyfroyt, Sem C. Borst, Onno J. Boxma, and Dee Denteneer. “A data propagation model for wireless gossiping”. In: *Performance Evaluation* 85–86.0 (2015), pp. 19 –32. ISSN: 0166-5316. DOI: http://dx.doi.org/10.1016/j.peva.2015.01.001.

[McN+14] Liam McNamara et al. “Demo Abstract: SicsthSense - Dispersing the Cloud”. In: *European Conference on Wireless Sensor Networks*. EWSN. 2014.

[MELT08] Razvan Musaloiu-Elefteri, Chieh-Jan Mike Liang, and Andreas Terzis. “Koala: Ultra-Low Power Data Retrieval in Wireless Sensor Networks”. In: *Symposium on Information Processing in Sensor Networks*. IPSN. 2008, pp. 421–432. DOI: 10.1109/IPSN.2008.10.

[MHS05] Raluca Marin Perianu, Pieter Hartel, and Hans Scholten. *A classification of service discovery protocols*. Tech. rep. TR-CTIT-05-25. Enschede, The Netherlands: University of Twente, 2005.

[MM12] Sebastian Marineau-Mes. *Why the internet of things needs ‘curated openness’*. Ed. by Mobilize Conference. [Online] https://gigaom.com/2012/09/21/rim-internet-of-things-mobilize-2012/. Sept. 2012.

[MP10] Chris Miller and Christian Poellabauer. “Reliable and efficient reprogramming in sensor networks”. In: *ACM Transactions on Sensor Networks* 7 (1 2010), 6:1–6:32. ISSN: 1550-4859. DOI: http://doi.acm.org/10.1145/1806895.1806901.

[MPSHH06] Raluca Marin-Perianu, Hans Scholten, Paul Havinga, and Pieter Hartel. “Energy-Efficient Cluster-Based Service Discovery in Wireless Sensor Networks”. In: *Conference on Local Computer Networks*. LCN. 2006, pp. 931 –938. DOI: 10.1109/LCN.2006.322202.

[MSL15] Thomas M.M. Meyfroyt, Milosh Stolikj, and Johan J. Lukkien. "Adaptive Broadcast Supression for Trickle-Based Protocols". In: *IEEE Symposium on A World of Wireless, Mobile and Multimedia Networks*. WoWMoM. June 2015, pp. 1–9. DOI: 10.1109/WoWMoM.2015.7158134.

[MTVDG13] Enzo Mingozzi, Giacomo Tanganelli, Carlo Vallati, and V. Di Gregorio. "An open framework for accessing Things as a service". In: *Symposium on Wireless Personal Multimedia Communications*. WPMC. June 2013, pp. 1–5.

[MV09] Francesco Marcelloni and Massimo Vecchio. "An Efficient Lossless Compression Algorithm for Tiny Nodes of Monitoring Wireless Sensor Networks". In: *The Computer Journal* 52.8 (2009), pp. 969–987.

[Nid01] Michael Nidd. "Service discovery in DEAPspace". In: *Personal Communications* 8.4 (2001), pp. 39–45. ISSN: 1070-9916. DOI: 10.1109/98.944002.

[OLBU10] Tanir Ozcelebi, Johan J. Lukkien, Remy Bosman, and Onder Uzun. "Discovery, monitoring and management in smart spaces composed of low capacity nodes". In: *IEEE Transactions on Consumer Electronics* 56.2 (2010), pp. 570–578. ISSN: 0098-3063.

[OPT13] George Oikonomou, Iain Phillips, and Theo Tryfonas. "IPv6 Multicast Forwarding in RPL-Based Wireless Sensor Networks". English. In: *Wireless Personal Communications* 73.3 (2013), pp. 1089–1116. ISSN: 0929-6212. DOI: 10.1007/s11277-013-1250-5.

[Ost+06] Frederik Osterlind et al. "Cross-Level Sensor Network Simulation with COOJA". In: *Conference on Local Computer Networks*. LCN. 2006, pp. 641–648. DOI: 10.1109/LCN.2006.322172.

[Pal+13] Maria Rita Palattella et al. "Standardized Protocol Stack for the Internet of (Important) Things". In: *IEEE Communications Surveys and Tutorials* 15.3 (2013), pp. 1389–1406. DOI: 10.1109/SURV.2012.111412.00158.

[PBM11] Rajesh Krishna Panta, Saurabh Bagchi, and Samuel P. Midkiff. "Efficient incremental code update for sensor networks". In: *ACM Transactions on Sensor Networks* 7 (4 2011), 30:1–30:32. ISSN: 1550-4859. DOI: 10.1145/1921621.1921624.

[Per03] Colin Percival. *Naive differences of executable code*. http://www.daemonology.net/bsdiff/. 2003.

[Pis97] Kristofer Pister. *Smart Dust*. Tech. rep. BAA97-43. University of Berkley, 1997.

[Pre+08] Alan Presser et al. *UPnP Device Architecture 1.1*. http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf. 2008.

[PSC05] Joseph Polastre, Robert Szewczyk, and David Culler. "Telos: Enabling Ultra-Low Power Wireless Research". In: *Symposium on Information Processing in Sensor Networks*. IPSN. 2005. ISBN: 0-7803-9202-7. DOI: 10.1109/IPSN.2005.1440950.

[RAYK10] Muhammad Taqi Raza, Fatima Muhammad Anwar, Seung-Wha Yoo, and Ki-Hyung Kim. "FESP: Fast and Energy Efficient Service Provisioning in 6LoWPAN". In: *Symposium on Personal Indoor and Mobile Radio Communications*. PIMRC. 2010, pp. 2575–2580. DOI: 10.1109/PIMRC.2010.5671763.

[RFC1034] Paul Mockapetris. *RFC 1034: Domain names - concepts and facilities*. http://www.ietf.org/rfc/rfc1034.txt. Internet Engineering Task Force, Nov. 1987.

[RFC1035] Paul Mockapetris. *RFC 1035: Domain names - implementation and specification*. http://www.ietf.org/rfc/rfc1035.txt. Internet Engineering Task Force, Nov. 1987.

[RFC1122] Robert Braden. *RFC 1122: Requirements for Internet Hosts - Communication Layers*. http://www.ietf.org/rfc/rfc1122.txt. Internet Engineering Task Force, Oct. 1989.

[RFC2460] Stephen Deering and Robert Hinden. *RFC 2460: Internet Protocol, Version 6 (IPv6) Specification*. http://www.ietf.org/rfc/rfc2460.txt. Internet Engineering Task Force, Dec. 1998.

[RFC2608] Erik Guttman, Charles Perkins, John Veizades, and Michael Day. *RFC 2608: Service Location Protocol, Version 2*. http://www.ietf.org/rfc/rfc2608.txt. Updated by RFC 3224. Internet Engineering Task Force, June 1999.

[RFC2608] Erik Guttman, Charles Perkins, John Veizades, and Michael Day. *Service Location Protocol, Version 2*. http://tools.ietf.org/rfc/rfc2608.txt. The Internet Society, 1999.

[RFC3284] David Korn, Joshua MacDonald, Jeffrey Mogul, and Kiem-Phong Vo. *RFC 3284: The VCDIFF Generic Differencing and Compression Data Format*. http://www.ietf.org/rfc/rfc3284.txt. Internet Engineering Task Force, June 2002.

[RFC4861] Thomas Narten, Erik Nordmark, William Allen Simpson, and Hesham Soliman. *RFC 4861: Neighbor Discovery for IP version 6 (IPv6)*. http://www.ietf.org/rfc/rfc4861.txt. Internet Engineering Task Force, Sept. 2007.

[RFC4919] Nandakishore Kushalnagar, Gabriel Montenegro, and Christian Schumacher. *RFC 4919: IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals*. http://www.ietf.org/rfc/rfc4919.txt. Internet Engineering Task Force, Aug. 2007.

[RFC4944] Gabriel Montenegro, Nandakishore Kushalnagar, Jonathan Hui, and Davidb Culler. *RFC 4944: Transmission of IPv6 Packets over IEEE 802.15.4 Networks*. http://www.ietf.org/rfc/rfc4944.txt. Updated by RFCs 6282, 6775. Internet Engineering Task Force, Sept. 2007.

[RFC6120] Peter Saint-Andre. *RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core*. http://www.ietf.org/rfc/rfc6120.txt. Internet Engineering Task Force, Mar. 2011.

[RFC6206] Philip Levis et al. *RFC 6206: The Trickle Algorithm*. http://www.ietf.org/rfc/rfc6206.txt. Internet Engineering Task Force, Mar. 2011.

[RFC6282] Jonathan Hui and Pascal Thubert. *RFC 6282: Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks*. http://www.ietf.org/rfc/rfc6282.txt. Internet Engineering Task Force, Sept. 2011.

[RFC6550] Tim Winter et al. *RFC 6550: RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*. http://www.ietf.org/rfc/rfc6550.txt. Internet Engineering Task Force, 2012.

[RFC6552] Pascal Thubert. *RFC 6552: Objective Function Zero for the Routing Protocol for Low-Power and Lossy Networks (RPL)*. http://www.ietf.org/rfc/rfc6552.txt. Internet Engineering Task Force, Mar. 2012.

[RFC6719] Omprakash Gnawali and Philip Levis. *RFC 6719: The Minimum Rank with Hysteresis Objective Function*. http://www.ietf.org/rfc/rfc6719.txt. Internet Engineering Task Force, Sept. 2012.

[RFC6762] Stuart Cheshire and Marc Krochmal. *RFC 6762: Multicast DNS*. http://www.ietf.org/rfc/rfc6762.txt. IETF, 2013.

[RFC6763] Stuart Cheshire and Marc Krochmal. *RFC 6763: DNS-Based Service Discovery*. http://www.ietf.org/rfc/rfc6763.txt. IETF, 2013.

[RFC6891] Joao Damas, Michael Graff, and Paul Vixie. *RFC 6891: Extension Mechanisms for DNS (EDNS(0))*. http://www.ietf.org/rfc/rfc6891.txt. Internet Engineering Task Force, Apr. 2013.

[RFC7049] Carsten Bormann and Paul Hoffman. *RFC 7049: Concise Binary Object Representation (CBOR)*. http://www.ietf.org/rfc/rfc7049.txt. Internet Engineering Task Force, Oct. 2012.

[RFC7159] Tim Bray. *RFC 7159: The JavaScript Object Notation (JSON) Data Interchange Format*. http://www.ietf.org/rfc/rfc7159.txt. Internet Engineering Task Force, Mar. 2014.

[RFC7228] Carsten Bormann, Mehmet Ersue, and Ari Keranen. *RFC 7228: Terminology for Constrained-Node Networks*. http://www.ietf.org/rfc/rfc7228.txt. Internet Engineering Task Force, May 2014.

[RL03] Niels Reijers and Koen Langendoen. "Efficient code distribution in wireless sensor networks". In: *Conference on Wireless sensor networks and applications*. WSNA. 2003, pp. 60–67. ISBN: 1-58113-764-8. DOI: http://doi.acm.org/10.1145/941350.941359.

[RW11] Nick Rozanski and Eóin Woods. *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives (2nd Edition)*. Addison-Wesley Professional, 2011. ISBN: 978-0321718334.

[SBR04] Gregor Schiele, Christian Becker, and Kurt Rothermel. "Energy-efficient cluster-based service discovery for Ubiquitous Computing". In: *Workshop on ACM SIGOPS*. EW 11. ACM, 2004.

[SC12] Nikolai Samteladze and Ken Christensen. "DELTA: Delta Encoding for Less Traffic for Apps". In: *Conference on Local Computer Networks*. LCN. Washington, DC, USA: IEEE Computer Society, 2012, pp. 212–215. ISBN: 978-1-4673-1565-4. DOI: 10.1109/LCN.2012.6423611.

[SCL12b] Milosh Stolikj, Pieter J.L. Cuijpers, and Johan J. Lukkien. "Energy-aware Reprogramming of Sensor Networks Using Incremental Update and Compression". In: *Procedia Computer Science* 10 (2012), pp. 179–187. ISSN: 1877-0509. DOI: 10.1016/j.procs.2012.06.026.

[SCNFR11] Bruno da Silva Campos, Eduardo Freire Nakamura, Carlos Mauricio S. Figueiredo, and Joel J.P.C. Rodrigues. "On the design of UPnP gateways for service discovery in wireless sensor networks". In: *IEEE Symposium on Computers and Communications*. ISCC. 2011, pp. 719–722. ISBN: 978-1-4577-0680-6.

[SCT09] Andy Stanford-Clark and Hong Linh Truong. *MQTT For Sensor Networks (MQTT-SN)*. http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf. IBM Corporation, 2009.

[SH11] Abolhassan Shamsaie and Jafar Habibi. “Planning updates in multi-application wireless sensor networks”. In: *Symposium on Computers and Communications*. ISCC. June 2011, pp. 802–808. DOI: 10.1109/ISCC.2011.5983940.

[SHB13] Zach Shelby, Klaus Hartke, and Carsten Bormann. *Constrained Application Protocol (CoAP)*. http://tools.ietf.org/html/draft-ietf-core-coap. Internet Engineering Task Force, 2013.

[SHE03] Thanos Stathopoulos, John Heidemann, and Deborah Estrin. *A remote code update mechanism for wireless sensor networks*. Tech. rep. DTIC Document, 2003.

[She11] Mark Shepard. *Sentient City: Ubiquitous Computing, Architecture, and the Future of Urban Space*. The MIT Press, 2011. ISBN: 0262515865, 9780262515863.

[SKPK14] John Schneider, Takuki Kamiya, Daniel Peintner, and Rumen Kyusakov. *W3C Recommendation: Efficient XML Interchange (EXI) Format*. http://www.w3.org/TR/exi/. World Wide Web Consorcium, 2014.

[SL12] Peter van der Stok and Kerry Lynn. *CoAP Utilization for Building Control*. http://tools.ietf.org/html/draft-vanderstok-core-bc. Internet Engineering Task Force, 2012.

[SLCB15] Milosh Stolikj, Johan J. Lukkien, Pieter J. L. Cuijpers, and Nina Buchina. “Nomadic Service Discovery in Smart Cities”. In: *Smart Cities and Homes: Key Enabling Technologies*. Ed. by Mohammad S. Obaidat and Petros Nicopolitidis. Elsevier, 2015.

[SM06] Christopher M. Sadler and Margaret Martonosi. “Data compression algorithms for energy-constrained devices in delay tolerant networks”. In: *Conference on Embedded networked sensor systems*. SenSys. Boulder, Colorado, USA, 2006, pp. 265–278. ISBN: 1-59593-343-3. DOI: 10.1145/1182807.1182834.

[SMCL15] Milosh Stolikj, Thomas M.M. Meyfroyt, Pieter J.L. Cuijpers, and Johan J. Lukkien. “Improving the Performance of Trickle-Based Data Dissemination in Low-Power Networks”. In: *European Conference on Wireless Sensor Networks*. EWSN. Feb. 2015, pp. 1–16. DOI: 10.1007/978-3-319-15582-1_12.

[Spi+09] Patrik Spiess et al. “SOA-Based Integration of the Internet of Things in Enterprise Services”. In: *IEEE International Conference on Web Services*. ICWS. July 2009, pp. 968–975. DOI: 10.1109/ICWS.2009.98.

[Sto+15] Peter van der Stok et al. *CoAP Management Interface*. http://tools.ietf.org/html/draft-vanderstok-core-comi. Internet Engineering Task Force, 2015.

[STS11] Jurgen Schoonwalder, Tina Tsou, and Behcet Sarikaya. “Protocol Profiles for Constrained Devices”. In: *Workshop on Interconnecting Smart Objects with the Internet*. 2011.

[Sub14] John Sublett. *Open to New Things*. Ed. by Oracle.com. [Online] http://www.oracle.com/us/corporate/profit/big-ideas/070914-jsublett-2244448.html. Aug. 2014.

[TA13] Internet of Things – Architecture. *Deliverable D1.5 – Final architectural reference model for the IoT v3.0*. [Online] http://www.iot-a.eu. 2013.

[TDHV09] Nicolas Tsiftes, Adam Dunkels, Zhitao He, and Thiemo Voigt. “Enabling Large-Scale Storage in Sensor Networks with the Coffee File System”. In: *Conference on Information Processing in Sensor Networks*. IPSN. 2009, pp. 349–360.

[TDV08] Nicola Tsiftes, Adam Dunkels, and Thiemo Voigt. “Efficient Sensor Network Reprogramming through Compression of Executable Modules”. In: *Conference on Sensor, Mesh and Ad Hoc Communications and Networks*. SECON. 2008, pp. 359–367. DOI: 10.1109/SAHCN.2008.51.

[TL09] Melissa Tjiong and Johan J. Lukkien. “On the False-Positive and False-Negative Behavior of a Soft-State Signaling Protocol”. In: *Conference on Advanced Information Networking and Applications*. AINA. 2009, pp. 971–979. DOI: 10.1109/AINA.2009.130.

[TOV10] Joydeep Tripathi, Jaudelice Cavalcante de Oliveira, and Jean-Philippe Vasseur. “A Performance Evaluation Study of RPL: Routing Protocol for Low power and Lossy Networks”. In: *Conference on Information Sciences and Systems*. CISS. 2010, pp. 1–6. DOI: 10.1109/CISS.2010.5464820.

[TPSBO09] Alessandra Toninelli, Susanna Pantsar-Syväniemi, Paolo Bellavista, and Eila Ovaska. “Supporting Context Awareness in Smart Environments: A Scalable Approach to Information Interoperability”. In: *Workhop on Middleware for Pervasive Mobile and Embedded Computing*. M-PAC. ACM, 2009, 5:1–5:4. ISBN: 978-1-60558-849-0. DOI: 10.1145/1657127.1657134.

[Tri00] Andrew Tridgell. *Efficient Algorithms for Sorting and Synchronization*. 2000.

[VD10] Jean-Philippe Vasseur and Adam Dunkels. “Chapter 1 - What Are Smart Objects?” In: *Interconnecting Smart Objects with IP*. Boston: Morgan Kaufmann, 2010, pp. 3 –20. ISBN: 978-0-12-375165-2. DOI: http://dx.doi.org/10.1016/B978-0-12-375165-2.00001-6.

[VM13] Carlo Vallati and Enzo Mingozzi. "Trickle-F: Fair Broadcast Suppression to Improve Energy-Efficient Route Formation with the RPL Routing Protocol". In: *Sustainable Internet and ICT for Sustainability*. SustainIT. 2013, pp. 1–9. DOI: 10.1109/SustainIT.2013.6685187.

[Wah14a] Peter Waher. *XEP-0322: Efficient XML Interchange (EXI) Format*. http://xmpp.org/extensions/xep-0322.html. XMPP Standards Foundation, 2014.

[Wah14b] Peter Waher. *XEP-0326: Internet of Things - Concentrators*. http://xmpp.org/extensions/xep-0326.html. XMPP Standards Foundation, 2014.

[Wah15a] Peter Waher. *XEP-0323: Internet of Things - Sensor Data*. http://xmpp.org/extensions/xep-0323.html. XMPP Standards Foundation, 2015.

[Wah15b] Peter Waher. *XEP-0324: Internet of Things - Provisioning*. http://xmpp.org/extensions/xep-0324.html. XMPP Standards Foundation, 2015.

[Wah15c] Peter Waher. *XEP-0325: Internet of Things - Control*. http://xmpp.org/extensions/xep-0325.html. XMPP Standards Foundation, 2015.

[Wal00] Jim Waldo. *The JINI Specifications*. 2nd. Addison-Wesley Longman Publishing, 2000. ISBN: 0201726173.

[Wan11] Roy Want. "Near Field Communication". In: *IEEE Pervasive Computing* 10.3 (2011), pp. 4–7. ISSN: 1536-1268. DOI: http://doi.ieeecomputersociety.org/10.1109/MPRV.2011.55.

[Wei14] Joseph Wei. "How Wearables Intersect with the Cloud and the Internet of Things : Considerations for the developers of wearables". In: *IEEE Consumer Electronics Magazine* 3.3 (July 2014), pp. 53–56. DOI: 10.1109/MCE.2014.2317895.

[WGB99] Mark Weiser, Rich Gold, and John Seely Brown. "The Origins of Ubiquitous Computing Research at PARC in the Late 1980s". In: *IBM Systems Journal* 38.4 (Dec. 1999), pp. 693–696. ISSN: 0018-8670. DOI: 10.1147/sj.384.0693.

[WK14] Peter Waher and Ronny Klauck. *XEP-0327: Internet of Things - Discovery*. http://xmpp.org/extensions/xep-0327.html. XMPP Standards Foundation, 2014.

[WKKK13] Honguk Woo, Hongsoo Kim, Kyusik Kim, and Dongkyoung Kim. "A large scale presence network for pervasive social computing". In: *Conference on Pervasive Computing and Communications Workshops*. PERCOM Workshops. Mar. 2013, pp. 145–150. DOI: 10.1109/PerComW.2013.6529472.

[YHE02] Wei Ye, J. Heidemann, and D. Estrin. “An energy-efficient MAC protocol for wireless sensor networks”. In: *Conference of the IEEE Computer and Communications Societies*. Vol. 3. INFOCOM. 2002, 1567–1576 vol.3. DOI: 10.1109/INFCOM.2002.1019408.

[YN13] Rayoung Yang and Mark W. Newman. “Learning from a Learning Thermostat: Lessons for Intelligent Systems for the Home”. In: *2013 ACM Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp. Zurich, Switzerland: ACM, 2013, pp. 93–102. ISBN: 978-1-4503-1770-2. DOI: 10.1145/2493432.2493489.

[Yoo15] BK Yoon. *The Internet of Things needs openness and industry collaboration to succeed*. Ed. by Samsung.com. [Online] http://www.samsung.com/us/news/newsRead.do?news_seq=24395. Jan. 2015.

[ZL77] Jacob Ziv and Abraham Lempel. “A universal algorithm for sequential data compression”. In: *IEEE Transactions on Information Theory* 23.3 (1977), pp. 337–343. ISSN: 0018-9448. DOI: 10.1109/TIT.1977.1055714.

[ZMN05] Feng Zhu, M.W. Mutka, and L.M. Ni. “Service discovery in pervasive computing environments”. In: *Pervasive Computing, IEEE* 4.4 (Oct. 2005), pp. 81–90. ISSN: 1536-1268. DOI: 10.1109/MPRV.2005.87.

[ZR13] Liang Zhou and Joel J. P. C. Rodrigues. “Service-oriented middleware for smart grid: Principle, infrastructure, and application”. In: *IEEE Communications Magazine* 51.1 (2013), pp. 84–89. DOI: 10.1109/MCOM.2013.6400443.

[ZSLM04] Pei Zhang, Christopher M. Sadler, Stephen A. Lyon, and Margaret Martonosi. “Hardware Design Experiences in ZebraNet”. In: *Conference on Embedded Networked Sensor Systems*. SenSys. 2004, pp. 227–238.

Acronyms

6LoWPAN IPv6 over LoWPAN.

AA Authoritative Answer.

ADMC Adjustable DNS Message Compression.

ARM Architectural Reference Model.

CBOR Concise Binary Object Representation.

CCA Clear-Channel Assessment.

CoAP Constrained Application Protocol.

CoMI CoAP Management Interface.

CSL Coordinated Sampled Listening.

CSMA/CA Carrier Sense Multiple Access with Collision Avoidance.

CTP Collection Tree Protocol.

DA Directory Agent.

DDNS Dynamic DNS Update.

DIO DODAG Information Object.

DIS DODAG Information Solicitation.

DNF Disjunctive Normal Form.

DNS Domain Name System.

DNS-SD DNS-Based Service Discovery.

DODAG Destination Oriented Directed Acyclic Graph.

DSL domain-specific language.

DTLS Datagram Transport Layer Security.

ENUM Electronic Number Mapping.

ETSI European Telecommunications Standards Institute.

ETX Expected Transmission Count.

EXI Efficient XML Interchange.

FastLZ Fast Lempel-Ziv.

FESP Fast and Energy Efficient Service Provisioning.

GL Group Leader.

HTTP Hyper Text Transfer Protocol.

IETF Internet Engineering Task Force.

IoT Internet of Things.

IOT-A Internet of Things - Architecture.

IP Internet Protocol.

IPv4 Internet Protocol version 4.

IPv6 Internet Protocol version 6.

JINI Java Intelligent Network Interface.

JSON JavaScript Object Notation.

LAN Local Area Network.

LLN Low-Power and Lossy Network.

LoWPAN Low-power Wireless Personal Area Network.

LZ77 Lempel-Ziv 77.

LZ78 Lempel-Ziv 78.

LZJB Lempel-Ziv-Jeff-Bonwick.

M2M Machine to Machine.

MAC Medium Access Control.

mDNS Multicast Domain Name System.

MPL Multicast Protocol for Low power and Lossy Networks.

MQTT Message Queuing Telemetry Transport.

MQTT-SN Message Queuing Telemetry Transport for Sensor Networks.

NFC Near-field Communication.

OASIS Organization for the Advancement of Structured Information Standards.

OF Objective Function.

OF0 Objective Function Zero.

OSAS Open Service Architecture for Sensors.

PLC Power Line Communication.

PS Proxy Server.

RAR Reference Architecture.

RDC Radio Duty Cycling.

REST Representational State Transfer.

RFC Request For Comments.

RFID Radio-Frequency Identification.

RLE Run-Length Encoding.

RM Resource Manager.

RMI Remote Method Invocation.

RMO Reference Model.

RPL IPv6 Routing Protocol for Low-Power and Lossy Networks.

RR Resource Record.

SA Service Agent.

SC Service Client.

SD Service Discovery.

SDP Service Discovery Protocol.

Sensor-LZW Sensor Lempel-Ziv-Welch.

SLP Service Location Protocol.

SOA Service Oriented Architecture.

SP Service Provider.

SR Service Repository.

SSDP Simple Service Discovery Protocol.

SSLP Simple Service Location Protocol.

SSP Sleeping Service Provider.

TA Translation Agent.

TCP Transmission Control Protocol.

UA User Agent.

UDGM Unit Disk Graph Radio Medium.

UDP User Datagram Protocol.

UPnP Universal Plug and Play.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

WSDL Web Services Description Language.

WSN Wireless Sensor Network.

XML Extensible Markup Language.

XMPP Extensible Messaging and Presence Protocol.

List of Figures

List of Tables

ACCOMPLISHMENTS

This list contains the publications which Milosh Stolikj co-authored during the writing of this thesis, Master theses he supervised and courses he was involved in.

Publications

Journals and book chapters

1. Milosh Stolikj, Johan J. Lukkien, Pieter J. L. Cuijpers, and Nina Buchina. "Nomadic Service Discovery in Smart Cities". In: *Smart Cities and Homes: Key Enabling Technologies*. Ed. by Mohammad S. Obaidat and Petros Nicopolitidis. Elsevier, 2015

2. Milosh Stolikj, Pieter J.L. Cuijpers, and Johan J. Lukkien. "Patching a patch - software updates using horizontal patching". In: *IEEE Transactions on Consumer Electronics* 59.2 (May 2013), pp. 435–441. ISSN: 0098-3063. DOI: 10.1109/TCE.2013.6531128

Conferences and workshops

1. Milosh Stolikj, Pieter J.L. Cuijpers, Johan J. Lukkien, and Nina Buchina. "Context Based Service Discovery in Unmanaged Networks Using mDNS/DNS-SD (Under review)". In: *IEEE Conference on Consumer Electronics*. ICCE. 2016

2. Milosh Stolikj, Thomas M.M. Meyfroyt, Pieter J.L. Cuijpers, and Johan J. Lukkien. "Improving the Performance of Trickle-Based Data Dissemination in Low-Power Networks". In: *European Conference on Wireless Sensor Networks*. EWSN. Feb. 2015, pp. 1–16. DOI: 10.1007/978-3-319-15582-1_12

3. Thomas M.M. Meyfroyt, Milosh Stolikj, and Johan J. Lukkien. "Adaptive Broadcast Supression for Trickle-Based Protocols". In: *IEEE Symposium on A World of Wireless, Mobile and Multimedia Networks*. WoWMoM. June 2015, pp. 1–9. DOI: 10.1109/WoWMoM.2015.7158134

4. Milosh Stolikj, Richard Verhoeven, Pieter J.L. Cuijpers, and Johan J. Lukkien. "Proxy support for service discovery using mDNS/DNS-SD in low power networks". In: *IEEE Symposium on A World of Wireless, Mobile and Multimedia Networks*. WoWMoM. June 2014, pp. 1–6. DOI: 10.1109/WoWMoM.2014.6918925

5. Milosh Stolikj, Pieter J.L. Cuijpers, and Johan J. Lukkien. "Efficient reprogramming of wireless sensor networks using incremental updates". In: *IEEE Conference on Pervasive Computing and Communications Workshops*. PERCOM Workshops. Mar. 2013, pp. 584–589. DOI: 10.1109/PerComW.2013.6529563

6. Milosh Stolikj, Pieter J.L. Cuijpers, and Johan J. Lukkien. “Patching a patch - software updates using horizontal patching”. In: *IEEE Transactions on Consumer Electronics* 59.2 (May 2013), pp. 435–441. ISSN: 0098-3063. DOI: 10.1109/TCE.2013.6531128

7. Milosh Stolikj, Pieter J.L. Cuijpers, and Johan J. Lukkien. “Patching a patch - software updates using horizontal patching”. In: *IEEE Conference on Consumer Electronics*. ICCE. 2013, pp. 647–648. DOI: 10.1109/ICCE.2013.6487054

8. Milosh Stolikj, Pieter J.L. Cuijpers, and Johan J. Lukkien. “Energy-aware Reprogramming of Sensor Networks Using Incremental Update and Compression”. In: *Procedia Computer Science* 10 (2012), pp. 179–187. ISSN: 1877-0509. DOI: 10.1016/j.procs.2012.06.026

Technical reports and project deliverables (non-refereed)

1. Milosh Stolikj, Pieter J. L. Cuijpers, and Johan J. Lukkien. *Deliverable 4.8: Design and Implementation of the Programming Framework*. COMMIT/ SenSafety, 2015

2. Julio Oliveira, Milosh Stolikj, Francesco Comaschi, and Maurits de Graaf. *Emergency communication technology for crowd safety*. [Online] http://www.commit-nl.nl/sites/default/files/23.%20Emergency%20communication%20technology%20for%20crowd%20safety.pdf. SenSafety, 2014

3. Milosh Stolikj, Thomas M.M. Meyfroyt, Pieter J.L. Cuijpers, and Johan J. Lukkien. *Improving the Performance of Trickle-Based Data Dissemination in Low-Power Networks*. Tech. rep. CS-14-10. Eindhoven University of Technology, 2014

4. Maurits de Graaf et al. *Deliverable 1.2: SenSafety Architecture Description*. COMMIT/ SenSafety, 2013

5. Milosh Stolikj, Pieter J. L. Cuijpers, and Johan J. Lukkien. *Efficient reprogramming of sensor networks using incremental updates and data compression*. Tech. rep. CS-12-10. Eindhoven University of Technology, 2012

6. Milosh Stolikj, Pieter J. L. Cuijpers, and Johan J. Lukkien. *Deliverable 4.6: Survey of Programming Frameworks*. COMMIT/ SenSafety, 2012

Education

Supervision

1. Nina Buchina. *Extending service discovery protocols with support for context information*. Master Thesis, Eindhoven University of Technology. 2014

Courses

1. **Internet of Things** for Professional Doctorate in Software Engineering (PDeng) trainees, year 2015/2016, Eindhoven University of Technology. Lectured by prof. dr. Johan J. Lukkien.

2. **Distributed Systems** for Bachelor students of Computer Science, year 2014/2015, Eindhoven University of Technology. Lectured by dr. Rudolf H. Mak.

3. **Operating systems** for Bachelor students of Computer Science, years 2010/2011 and 2012/2013, Eindhoven University of Technology. Lectured by dr. Tanir Ozcelebi.

IPA Dissertation Series

Titles in the IPA Dissertation Series since 2009

M.H.G. Verhoef. *Modeling and Validating Distributed Embedded Real-Time Control Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2009-01

M. de Mol. *Reasoning about Functional Programs: Sparkle, a proof assistant for Clean*. Faculty of Science, Mathematics and Computer Science, RU. 2009-02

M. Lormans. *Managing Requirements Evolution*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-03

M.P.W.J. van Osch. *Automated Model-based Testing of Hybrid Systems*. Faculty of Mathematics and Computer Science, TU/e. 2009-04

H. Sozer. *Architecting Fault-Tolerant Software Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-05

M.J. van Weerdenburg. *Efficient Rewriting Techniques*. Faculty of Mathematics and Computer Science, TU/e. 2009-06

H.H. Hansen. *Coalgebraic Modelling: Applications in Automata Theory and Modal Logic*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-07

A. Mesbah. *Analysis and Testing of Ajax-based Single-page Web Applications*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-08

A.L. Rodriguez Yakushev. *Towards Getting Generic Programming Ready for Prime Time*. Faculty of Science, UU. 2009-9

K.R. Olmos Joffré. *Strategies for Context Sensitive Program Transformation*. Faculty of Science, UU. 2009-10

J.A.G.M. van den Berg. *Reasoning about Java programs in PVS using JML*. Faculty of Science, Mathematics and Computer Science, RU. 2009-11

M.G. Khatib. *MEMS-Based Storage Devices. Integration in Energy-Constrained Mobile Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-12

S.G.M. Cornelissen. *Evaluating Dynamic Analysis Techniques for Program Comprehension*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2009-13

D. Bolzoni. *Revisiting Anomaly-based Network Intrusion Detection Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-14

H.L. Jonker. *Security Matters: Privacy in Voting and Fairness in Digital Exchange*. Faculty of Mathematics and Computer Science, TU/e. 2009-15

M.R. Czenko. *TuLiP - Reshaping Trust Management*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-16

T. Chen. *Clocks, Dice and Processes*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2009-17

C. Kaliszyk. *Correctness and Availability: Building Computer Algebra on top of Proof Assistants and making Proof Assistants available over the Web*. Faculty of Science, Mathematics and Computer Science, RU. 2009-18

R.S.S. O'Connor. *Incompleteness & Completeness: Formalizing Logic and Analysis in Type Theory*. Faculty of Science, Mathematics and Computer Science, RU. 2009-19

B. Ploeger. *Improved Verification Methods for Concurrent Systems*. Faculty of Mathematics and Computer Science, TU/e. 2009-20

T. Han. *Diagnosis, Synthesis and Analysis of Probabilistic Models*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-21

R. Li. *Mixed-Integer Evolution Strategies for Parameter Optimization and Their Applications to Medical Image Analysis*. Faculty of Mathematics and Natural Sciences, UL. 2009-22

J.H.P. Kwisthout. *The Computational Complexity of Probabilistic Networks*. Faculty of Science, UU. 2009-23

T.K. Cocx. *Algorithmic Tools for Data-Oriented Law Enforcement*. Faculty of Mathematics and Natural Sciences, UL. 2009-24

A.I. Baars. *Embedded Compilers*. Faculty of Science, UU. 2009-25

M.A.C. Dekker. *Flexible Access Control for Dynamic Collaborative Environments*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2009-26

J.F.J. Laros. *Metrics and Visualisation for Crime Analysis and Genomics*. Faculty of Mathematics and Natural Sciences, UL. 2009-27

C.J. Boogerd. *Focusing Automatic Code Inspections*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2010-01

M.R. Neuhäußer. *Model Checking Nondeterministic and Randomly Timed Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-02

J. Endrullis. *Termination and Productivity*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-03

T. Staijen. *Graph-Based Specification and Verification for Aspect-Oriented Languages*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2010-04

Y. Wang. *Epistemic Modelling and Protocol Dynamics*. Faculty of Science, UvA. 2010-05

J.K. Berendsen. *Abstraction, Prices and Probability in Model Checking Timed Automata*. Faculty of Science, Mathematics and Computer Science, RU. 2010-06

A. Nugroho. *The Effects of UML Modeling on the Quality of Software*. Faculty of Mathematics and Natural Sciences, UL. 2010-07 **A. Silva**. *Kleene Coalgebra*.

Faculty of Science, Mathematics and Computer Science, RU. 2010-08

J.S. de Bruin. *Service-Oriented Discovery of Knowledge - Foundations, Implementations and Applications*. Faculty of Mathematics and Natural Sciences, UL. 2010-09

D. Costa. *Formal Models for Component Connectors*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2010-10

M.M. Jaghoori. *Time at Your Service: Schedulability Analysis of Real-Time and Distributed Services*. Faculty of Mathematics and Natural Sciences, UL. 2010-11

R. Bakhshi. *Gossiping Models: Formal Analysis of Epidemic Protocols*. Faculty of Sciences, Department of Computer Science, VUA. 2011-01

B.J. Arnoldus. *An Illumination of the Template Enigma: Software Code Generation with Templates*. Faculty of Mathematics and Computer Science, TU/e. 2011-02

E. Zambon. *Towards Optimal IT Availability Planning: Methods and Tools*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-03

L. Astefanoaei. *An Executable Theory of Multi-Agent Systems Refinement*. Faculty of Mathematics and Natural Sciences, UL. 2011-04

J. Proença. *Synchronous coordination of distributed components*. Faculty of Mathematics and Natural Sciences, UL. 2011-05

A. Moralı. *IT Architecture-Based Confidentiality Risk Assessment in Networks of Organizations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-06

M. van der Bijl. *On changing models in Model-Based Testing*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-07

C. Krause. *Reconfigurable Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-08

M.E. Andrés. *Quantitative Analysis of Information Leakage in Probabilistic and Nondeterministic Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2011-09

M. Atif. *Formal Modeling and Verification of Distributed Failure Detectors*. Faculty of Mathematics and Computer Science, TU/e. 2011-10

P.J.A. van Tilburg. *From Computability to Executability – A process-theoretic view on automata theory*. Faculty of Mathematics and Computer Science, TU/e. 2011-11

Z. Protic. *Configuration management for models: Generic methods for model comparison and model co-evolution*. Faculty of Mathematics and Computer Science, TU/e. 2011-12

S. Georgievska. *Probability and Hiding in Concurrent Processes*. Faculty of Mathematics and Computer Science, TU/e. 2011-13

S. Malakuti. *Event Composition Model: Achieving Naturalness in Runtime Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2011-14

M. Raffelsieper. *Cell Libraries and Verification*. Faculty of Mathematics and Computer Science, TU/e. 2011-15

C.P. Tsirogiannis. *Analysis of Flow and Visibility on Triangulated Terrains*. Faculty of Mathematics and Computer Science, TU/e. 2011-16

Y.-J. Moon. *Stochastic Models for Quality of Service of Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-17

R. Middelkoop. *Capturing and Exploiting Abstract Views of States in OO Verification*.

Faculty of Mathematics and Computer Science, TU/e. 2011-18

M.F. van Amstel. *Assessing and Improving the Quality of Model Transformations*. Faculty of Mathematics and Computer Science, TU/e. 2011-19

A.N. Tamalet. *Towards Correct Programs in Practice*. Faculty of Science, Mathematics and Computer Science, RU. 2011-20

H.J.S. Basten. *Ambiguity Detection for Programming Language Grammars*. Faculty of Science, UvA. 2011-21

M. Izadi. *Model Checking of Component Connectors*. Faculty of Mathematics and Natural Sciences, UL. 2011-22

L.C.L. Kats. *Building Blocks for Language Workbenches*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2011-23 **S. Kemper**. *Modelling and Analysis of Real-Time Coordination Patterns*. Faculty of Mathematics and Natural Sciences, UL. 2011-24

J. Wang. *Spiking Neural P Systems*. Faculty of Mathematics and Natural Sciences, UL. 2011-25

A. Khosravi. *Optimal Geometric Data Structures*. Faculty of Mathematics and Computer Science, TU/e. 2012-01

A. Middelkoop. *Inference of Program Properties with Attribute Grammars, Revisited*. Faculty of Science, UU. 2012-02

Z. Hemel. *Methods and Techniques for the Design and Implementation of Domain-Specific Languages*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-03

T. Dimkov. *Alignment of Organizational Security Policies: Theory and Practice*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-04

S. Sedghi. *Towards Provably Secure Efficiently Searchable Encryption*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-05

F. Heidarian Dehkordi. *Studies on Verification of Wireless Sensor Networks and Abstraction Learning for System Inference*. Faculty of Science, Mathematics and Computer Science, RU. 2012-06

K. Verbeek. *Algorithms for Cartographic Visualization*. Faculty of Mathematics and Computer Science, TU/e. 2012-07

D.E. Nadales Agut. *A Compositional Interchange Format for Hybrid Systems: Design and Implementation*. Faculty of Mechanical Engineering, TU/e. 2012-08

H. Rahmani. *Analysis of Protein-Protein Interaction Networks by Means of Annotated Graph Mining Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2012-09

S.D. Vermolen. *Software Language Evolution*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2012-10

L.J.P. Engelen. *From Napkin Sketches to Reliable Software*. Faculty of Mathematics and Computer Science, TU/e. 2012-11

F.P.M. Stappers. *Bridging Formal Models – An Engineering Perspective*. Faculty of Mathematics and Computer Science, TU/e. 2012-12

W. Heijstek. *Software Architecture Design in Global and Model-Centric Software Development*. Faculty of Mathematics and Natural Sciences, UL. 2012-13 **C. Kop**.

Higher Order Termination. Faculty of Sciences, Department of Computer Science, VUA. 2012-14

A. Osaiweran. *Formal Development of Control Software in the Medical Systems Domain*. Faculty of Mathematics and Computer Science, TU/e. 2012-15

W. Kuijper. *Compositional Synthesis of Safety Controllers*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2012-16

H. Beohar. *Refinement of Communication and States in Models of Embedded Systems*. Faculty of Mathematics and Computer Science, TU/e. 2013-01

G. Igna. *Performance Analysis of Real-Time Task Systems using Timed Automata*. Faculty of Science, Mathematics and Computer Science, RU. 2013-02

E. Zambon. *Abstract Graph Transformation – Theory and Practice*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2013-03

B. Lijnse. *TOP to the Rescue – Task-Oriented Programming for Incident Response Applications*. Faculty of Science, Mathematics and Computer Science, RU. 2013-04

G.T. de Koning Gans. *Outsmarting Smart Cards*. Faculty of Science, Mathematics and Computer Science, RU. 2013-05

M.S. Greiler. *Test Suite Comprehension for Modular and Dynamic Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2013-06

L.E. Mamane. *Interactive mathematical documents: creation and presentation*. Faculty of Science, Mathematics and Computer Science, RU. 2013-07

M.M.H.P. van den Heuvel. *Composition and synchronization of real-time components upon one processor*. Faculty of Mathematics and Computer Science, TU/e. 2013-08

J. Businge. *Co-evolution of the Eclipse Framework and its Third-party Plug-ins*. Faculty of Mathematics and Computer Science, TU/e. 2013-09

S. van der Burg. *A Reference Architecture for Distributed Software Deployment*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2013-10

J.J.A. Keiren. *Advanced Reduction Techniques for Model Checking*. Faculty of Mathematics and Computer Science, TU/e. 2013-11

D.H.P. Gerrits. *Pushing and Pulling: Computing push plans for disk-shaped robots, and dynamic labelings for moving points*. Faculty of Mathematics and Computer Science, TU/e. 2013-12

M. Timmer. *Efficient Modelling, Generation and Analysis of Markov Automata*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2013-13

M.J.M. Roeloffzen. *Kinetic Data Structures in the Black-Box Model*. Faculty of Mathematics and Computer Science, TU/e. 2013-14

L. Lensink. *Applying Formal Methods in Software Development*. Faculty of Science, Mathematics and Computer Science, RU. 2013-15

C. Tankink. *Documentation and Formal Mathematics — Web Technology meets Proof Assistants*. Faculty of Science, Mathematics and Computer Science, RU. 2013-16

C. de Gouw. *Combining Monitoring with Run-time Assertion Checking*. Faculty of Mathematics and Natural Sciences, UL. 2013-17

J. van den Bos. *Gathering Evidence: Model-Driven Software Engineering in Automated Digital Forensics*. Faculty of Science, UvA. 2014-01

D. Hadziosmanovic. *The Process Matters: Cyber Security in Industrial Control Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-02

A.J.P. Jeckmans. *Cryptographically-Enhanced Privacy for Recommender Systems*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-03

C.-P. Bezemer. *Performance Optimization of Multi-Tenant Software Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2014-04

T.M. Ngo. *Qualitative and Quantitative Information Flow Analysis for Multi-threaded Programs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-05

A.W. Laarman. *Scalable Multi-Core Model Checking*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-06

J. Winter. *Coalgebraic Characterizations of Automata-Theoretic Classes*. Faculty of Science, Mathematics and Computer Science, RU. 2014-07

W. Meulemans. *Similarity Measures and Algorithms for Cartographic Schematization*. Faculty of Mathematics and Computer Science, TU/e. 2014-08

A.F.E. Belinfante. *JTorX: Exploring Model-Based Testing*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2014-09

A.P. van der Meer. *Domain Specific Languages and their Type Systems*. Faculty of Mathematics and Computer Science, TU/e. 2014-10

B.N. Vasilescu. *Social Aspects of Collaboration in Online Software Communities*. Faculty of Mathematics and Computer Science, TU/e. 2014-11

F.D. Aarts. *Tomte: Bridging the Gap between Active Learning and Real-World Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2014-12

N. Noroozi. *Improving Input-Output Conformance Testing Theories*. Faculty of Mathematics and Computer Science, TU/e. 2014-13 **M. Helvensteijn**. *Abstract Delta Modeling: Software Product Lines and Beyond*. Faculty of Mathematics and Natural Sciences, UL. 2014-14

P. Vullers. *Efficient Implementations of Attribute-based Credentials on Smart Cards*. Faculty of Science, Mathematics and Computer Science, RU. 2014-15

F.W. Takes. *Algorithms for Analyzing and Mining Real-World Graphs*. Faculty of Mathematics and Natural Sciences, UL. 2014-16

M.P. Schraagen. *Aspects of Record Linkage*. Faculty of Mathematics and Natural Sciences, UL. 2014-17

G. Alpár. *Attribute-Based Identity Management: Bridging the Cryptographic Design of ABCs with the Real World*. Faculty of Science, Mathematics and Computer Science, RU. 2015-01

A.J. van der Ploeg. *Efficient Abstractions for Visualization and Interaction*. Faculty of Science, UvA. 2015-02

R.J.M. Theunissen. *Supervisory Control in Health Care Systems*. Faculty of Mechanical Engineering, TU/e. 2015-03

T.V. Bui. *A Software Architecture for Body Area Sensor Networks: Flexibility and Trustworthiness*. Faculty of Mathematics and Computer Science, TU/e. 2015-04

A. Guzzi. *Supporting Developers' Teamwork from within the IDE*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-05

T. Espinha. *Web Service Growing Pains: Understanding Services and Their Clients*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-06

S. Dietzel. *Resilient In-network Aggregation for Vehicular Networks*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-07

E. Costante. *Privacy throughout the Data Cycle*. Faculty of Mathematics and Computer Science, TU/e. 2015-08

S. Cranen. *Getting the point — Obtaining and understanding fixpoints in model checking*. Faculty of Mathematics and Computer Science, TU/e. 2015-09

R. Verdult. *The (in)security of proprietary cryptography*. Faculty of Science, Mathematics and Computer Science, RU. 2015-10

J.E.J. de Ruiter. *Lessons learned in the analysis of the EMV and TLS security protocols*. Faculty of Science, Mathematics and Computer Science, RU. 2015-11

Y. Dajsuren. *On the Design of an Architecture Framework and Quality Evaluation for Automotive Software Systems*. Faculty of Mathematics and Computer Science, TU/e. 2015-12

J. Bransen. *On the Incremental Evaluation of Higher-Order Attribute Grammars*. Faculty of Science, UU. 2015-13

S. Picek. *Applications of Evolutionary Computation to Cryptology*. Faculty of Science, Mathematics and Computer Science, RU. 2015-14

C. Chen. *Automated Fault Localization for Service-Oriented Software Systems*. Faculty of Electrical Engineering, Mathematics, and Computer Science, TUD. 2015-15

S. te Brinke. *Developing Energy-Aware Software*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2015-16

R.W.J. Kersten. *Software Analysis Methods for Resource-Sensitive Systems*. Faculty of Science, Mathematics and Computer Science, RU. 2015-17

J.C. Rot. *Enhanced coinduction*. Faculty of Mathematics and Natural Sciences, UL. 2015-18

M. Stolikj. *Building Blocks for the Internet of Things*. Faculty of Mathematics and Computer Science, TU/e. 2015-19

SUMMARY

The Internet of Things (IoT) is a vision where every physical object is connected to the Internet, and is able to uniquely identify itself to other devices. The interconnection of these so-called smart objects enables a plethora of new use cases, such as Smart Grids, Smart Cities, Industrial Automation, Home Automation, Building Automation etc. Several sources estimate that, by 2020, the IoT will bring together between 20 and 30 billion devices, with varying degrees of complexity. The sheer size of the IoT, and the expected heterogeneity, pose a challenge to the current state of the art software protocols.

The research community has been actively working on finding appropriate solutions for the IoT. The communities' stance is to build the network around standards, which would enable easier integration of heterogeneous devices. However, many of the protocols involved are under development, or difficult to implement in the low end part of the IoT, where devices are often battery powered, communication links are lossy, and there is not enough processing capacity.

The aim of this dissertation is to improve the state of the art in protocol design for the IoT. The scientific contributions of the dissertation apply to the application, network and link layers in low power, lossy networks of resource constrained devices (LLNs).

The thesis starts with an overview of the current landscape of IoT issues and solutions. We define a service oriented architecture on top of open standards, which incorporates resource constrained devices. From the proposed architecture and the existing IoT solutions, we identify three research areas, which are treated in the subsequent chapters.

Firstly, we focus on improving software updates in large LLNs. Delivering such updates is slow and energy draining, due to the low link capacity and the size of the software updates themselves. To resolve this issue, we investigate how data compression and incremental updates algorithms can be exploited in LLNs. We provide further improvements for reducing the size of updates when multiple similar, but not identical devices in the same network need to updated. Our results show up to 70% reduction in the size of the updates, compared to traditional data compression algorithms.

Secondly, we investigate service discovery protocols for LLNs. Existing protocols, such as Multicast DNS with DNS-Based Service Discovery (DNS-SD), are inapplicable to LLNs due to their always-on requirements and heavy traffic load. To address these issues, we propose two extensions of the protocol. The first extension introduces proxy servers, which take over service discovery responsibilities from resource constrained devices. The second extension enhances service selection criteria, by enabling devices to be looked up based on their physical properties, such as location, available sensors etc.

Thirdly, we analyze the network layer of low power devices, and its interplay with the link layer. Two fundamental IoT protocols, for routing (RPL) and multicast forwarding (MPL), rely on the Trickle protocol for fast data dissemination with little redundant traffic. However, we show that Trickle is not resilient to varying network topology, and due to the lack of feedback from the link layer, can suffer from starvation, sub-optimal routing and excessive overhead traffic. To resolve these issues, we propose extensions to both the Trickle protocol, which makes it scalable in networks with varying density, as well the link layer, which prevents starvation to occur.

In summary, the contributions presented in this thesis are a step towards the implementation of a fully connected world, where each device is directly accessible, and open for interaction.

CURRICULUM VITAE

Milosh Stolikj was born on 28-02-1984 in Skopje, Macedonia. He finished Bachelor and Master studies at the Institute of Informatics, Ss. Cyril and Methodius University in Skopje, Macedonia, in 2006 and 2010, respectively.

He started working as a part time teaching assistant at the Ss. Cyril and Methodius University, Skopje, Macedonia, from 2005 to 2009. At the same period, he worked as a visiting teaching assistant at New York University, Skopje, Macedonia. Between 2006 and 2007, he worked as a software engineer in Maksat, Skopje, Macedonia. Next, from 2007 to 2009, he worked as a senior software engineer at Cabletel, Skopje, Macedonia. Then, from 2009 to 2011, he worked as a research assistant at the Ss. Cyril and Methodius University, Skopje, Macedonia.

In June 2011 he joined the System Architecture and Networking group at the Mathematics and Computer Science Department, Eindhoven University of Technology, as a PhD candidate. The results from this work are presented in this thesis.