FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Scalability Analysis of a Web-based IoT Stack for Automation Systems

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Philipp Sebastian Raich
Matrikelnummer 0404014

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Mitwirkung: Projektass. Dipl.-Ing. Markus Helmut Jung

Wien, 16.01.2015

(Unterschrift Philipp Sebastian Raich)

(Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Scalability Analysis of a Web-based IoT Stack for Automation Systems

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Philipp Sebastian Raich
Registration Number 0404014

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Assistance: Projektass. Dipl.-Ing. Markus Helmut Jung

Vienna, 16.01.2015

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Philipp Sebastian Raich
Gentzgasse 160/2/13, 1180 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Philipp Sebastian Raich)

Acknowledgements

I owe a big Thank You to my supervisors, Wolfgang Kastner and foremost Markus Jung, for their persevering support, their helpful feedback, their attention to detail and their patience. They are known for their tireless work ethic and being able to witness it first hand was inspiring. This work would not have been possible without them. A bright future awaits Markus, and I wish him all the best.

My parents, Linda and Siegfried, I cannot thank enough in written words for their support over the last 28 years. I owe them everything I have, and everything I am. Their patience with me and their belief in me are unimaginable and admirable and their constant effort to motivate me, made it possible for me to be where I am now, and I am still catching up on what tried to teach me. Their lessons, pacifications and lullabies will always accompany me wherever I go. My grandparents will never know how much of them is in me, and how much I learned from them in the little time we had. I sincerely hope I can one day give my grandchildren half as much as my grandparents gave me.

I am also very grateful for my former flat mates, Florian and Johannes, which accompanied and sustained me and my attitude, mishaps and temper throughout most of my academic career. Both are very intelligent and sensitive persons, and I see a bright and happy future for them. For the longest time, they were my family and my safety net, and I hope I can still call them my friends and companions in the far future, even if we don't share coffee regularly anymore.

Additionally, I would like to thank my colleagues at university I had the joy to work, learn and have fun with. Andreas helped me through most of my Bachelor's degree, reminding me to keep working, guiding me, and showing me that the things I don't know might be hard, but not black magic. Kyrill was an inspiration, intelligent and persistent, yet down to earth, patient and never arrogant. It was also great fun working with Eugen, Frieder, Matthias and Hubert on those assignments, as they made it a lot easier and fun for me to catch up in my new field.

My dear friend, Thomas, I want to thank for his patience while listening to me and his constant effort to encourage me. Being a curious person and a smart mind, he is most certainly the prototype of a researcher, but most of all, he is a kind and caring person, reasonable and wise, yet youthful and ludicrous. His belief in me still motivates me to this day, and I enjoy every moment we have, to find the best coffee or beer on this side of the earth.

Most of all I want to thank my partner and love, Kerstin, for her support and encouragement, as well as for giving me comfort and care and most of all her patience with me. She will always be the better one of us two, keeping things in place, and her patience, dedication and joyfulness are an inspiration. I only hope she will one day understand how smart she really is, and believe as much in herself as she always believes in me.

Abstract

The *Internet of Things (IoT)* is commonly anticipated to become an omnipresent reality in the near future. One self-evident application is the integration of wireless, constrained devices (e.g. sensors and actuators for building automation) and their respective Wireless Sensor and Actuator Networks (WSANs) into the Internet, and hence making them generally accessible for interaction with other WSANs and other Internet enabled devices. Based on considerable efforts taken to successfully establish standardized network layers for this purpose, a common application protocol is still required to guarantee interoperability. This work proposes the use of *Constrained Application Protocol (CoAP)* as a standard for interactions between and with constrained devices, which was recently promoted into Request for Comments (RFC) status by the Constrained RESTful Environments (CoRE) working group of the Internet Engineering Task Force (IETF). Further, the *Open Building Information Exchange (OBIX)* interface specification shall be used to represent interaction points provided by the respective devices in a standardized way. Establishing this stack on constrained devices might solve the interoperability issues the IoT is currently up against.

The question remains if such a solution with its proposed components is generally feasible and if crucial metrics (e.g. energy consumption) are within acceptable ranges. In particular, questions regarding the binary footprint of the solution and its capability for efficient group communication are yet unsettled. This work verifies the feasibility of the proposed stack through a *proof of concept* implementation, and thus tries to uncover shortcomings and remaining practical limits. Further, *simulations* are run to analyze the stack and its behavior in detail, under certain scenarios and with different simulation parameters.

This work presents an implementation of the aforementioned stack on constrained devices, which can interact with other CoAP-enabled devices outside its WSAN. Further, a simulation framework is proposed, which is used to run various simulations regarding scalability of the stack. A group communication facility is presented, to demonstrate and evaluate group communication with CoAP devices and multicast routing engines for the respective networks. The results lead to a better understanding of the practical aspects of the proposed solution, where its limits currently are and which questions are yet to be solved.

Kurzfassung

Das *Internet of Things (IoT)* (dt. “Internet der Dinge”) wird von der wissenschaftlichen Gemeinschaft als eine gegebene, unmittelbare Entwicklung angesehen. Neben anderen, ist eine naheliegende Anwendung die Integration von drahtlosen, eingeschränkten Geräten (z.B. Sensoren und Aktuatoren für die Gebäudeautomation) und deren drahtlose Netzwerke in das Internet, um sie so untereinander und mit Geräten in anderen Netzwerken/dem Internet zu verknüpfen. Basierend auf beachtlichen Fortschritten auf dem Gebiete der Standardisierung von passenden Netzwerkprotokollen wird ein gemeinsames Applikationsprotokoll benötigt, um Interoperabilität zu garantieren. Im Folgenden wird dafür die Verwendung von *Constrained Application Protocol (CoAP)* vorgeschlagen, das unlängst von der Internet Engineering Task Force (IETF) als Standard für die Kommunikation mit und zwischen eingeschränkten Geräten in den Request for Comments (RFC)-Status erhoben wurde. Desweiteren soll *Open Building Information Exchange (OBIX)* als Spezifikation für eine standardisierte Datenpunkt-Representation dienen. Das Etablieren dieses Protokollstapels auf eingeschränkten Geräten könnte zahlreiche Interoperabilitätsprobleme lösen, gegen die das IoT derzeit ankommen muss.

Offen ist die Frage, ob diese Lösung mit allen ihren Bausteinen machbar ist und wie sich darin bestimmte Metriken (z.B. Energieverbrauch) verhalten. Im Besonderen stellen sich die Fragen, ob der Protokollstapel als solcher auf Grund des begrenzten Speichers auf einem eingeschränkten Gerät implementieren und einsetzen lässt, und wie sich damit effiziente Gruppenkommunikation bewerkstelligen lässt. Diese Arbeit weist die Machbarkeit der vorgeschlagenen Lösung mit einer *praktischen Machbarkeitsstudie* nach und versucht dabei Schwachstellen und ungeklärte Fragen aufzudecken. Zusätzlich wird die Lösung einer Reihe von *Simulationen* unterzogen, um weitere Details zum Verhalten in unterschiedlichen Szenarien und unter unterschiedlichen Bedingungen zu erforschen.

Im Zuge dieser Arbeit wird eine Umsetzung des bereits erwähnten Protokollstapels auf eingeschränkten Geräten vorgestellt, die mit anderen CoAP-Geräten außerhalb des eigenen Netzwerks kommunizieren können. Um die Lösung eingehend auf ihre Skalierbarkeit hin zu untersuchen, wird zusätzlich eine automatisierte Simulationsumgebung präsentiert. Darüber hinaus werden Änderungen gezeigt, die Gruppenkommunikation auf CoAP-Geräten möglich machen, um somit die Bestandteile und Effekte effizienter Gruppenkommunikation untersuchen zu können. Die präsentierten Ergebnisse ermöglichen ein besseres Verständnis der praktischen Aspekte der vorgeschlagenen Lösung, wo deren Grenzen liegen und welche fortführenden Untersuchungen noch notwendig sind.

Contents

CHAPTER 1

Introduction

1.1 The "Internet of Things"

The term "Internet of Things" has become a synonym for various different ideas that have one thing in common: The integration of devices of all possible kinds into the Internet, making them not only generally accessible for human interaction, but also interconnecting them, thus enabling all sorts of Machine to Machine (M2M) applications.

Due to its nature, the Internet of Things (IoT) as such is not limited to any of the synonymous use cases, applications, or technologies. Nevertheless, it is not necessarily but in most of its iterations tightly coupled to the efforts to deploy IPv6 not only on devices already connected to the Internet, but also on devices that until now were not considered predestined as participants of the Internet, e.g. home appliances, wearables, cars, or even arbitrary items. Most importantly, a completely new class of Internet devices is supposed to emerge with the IoT: **small, light, self powered, low-cost, wireless devices**, that enable completely new applications by themselves, e.g. acting as independent sensor nodes or wireless agents. We will further call these devices "constrained devices".

Unfortunately, there are as much solutions to the IoT as there are ideas and protagonists, and thus at least as many devices, standards and application protocols emerged so far [1], sometimes proprietary, ensuring incompatibility and thus undermining on of the core ideas of the IoT, **ubiquity through interoperability**. This means that currently we are yet far away from having the comfort and convenience the IoT was supposed to bring into our lives [2], and still preoccupied with integration of devices that use foreign standards.

To reach the goal of a device independent, ubiquitous IoT, **a common, open application layer** based on open standards must be defined, that further can be placed on top of established and open and standardized protocols. In addition to the common application layer, a **common and open interface and data representation** is necessary, ensuring that devices can use the common application layer to exchange information.

1.2 Use Cases for the IoT

All the possible use cases for the IoT in their entirety are impossible to grasp, especially if the history of the Internet is taken as representative example and how most of today's uses of the Internet would have been unimaginable at the time it was conceived, instead they emerged from the possibilities opened by the Internet itself.

Nevertheless, several use cases are either anticipated or already discernible and will thus be mentioned.

1.2.1 Wearables

The rise of wearable devices, also called "wearables", i.e. electronic devices of any kind that are worn on the body or with clothing, is already underway and accelerating. As for now, wearables are primarily wireless smart devices, that are worn by the user throughout the day and offer continuous collection of various sensor data, or act as remote for other devices in their direct vicinity. Through the integration with other devices and/or remote services, these devices and their data are available for more advanced uses, as activity and health monitoring, notification display, phone call handling, etc. In the near future, with the respective technological advancements, these devices might outgrow their compulsory Internet enabled companions, and further eventually replace them. This would include that they become capable to join the Internet without accessory device/gateway and in consequence that they are accessible for other devices through the Internet. At the same time radically smaller devices, that can be invisibly embedded in our clothing, might emerge, and which still might use nearby devices as gateway for Internet access for the sake of reduced power consumption. Nevertheless, following the idea of the IoT, these devices will most certainly not be "hidden" behind their gateways like their present incarnations, but be **native Internet citizens**, which are fully integrated into the Internet and thus reachable from every other Internet device via their IP address.

1.2.2 Home Appliances

Smart, Internet-enabled home appliances was one of the first anticipated use cases for the IoT. The idea is that the single purpose devices we currently have in our homes record information about their use, their operation, their contents or even their users, and use the data to offer it to the users or build new services with that information, e.g. the fridge that keeps track of its content and reminds the user to buy milk if its inventory is depleted or expired, or a toilet that monitors some of the user's health parameters and recommends dietary options or appointments with a physician accordingly. The capabilities of everyday objects could be drastically increased, making them remotely controllable, and further enable them to initiate communication with the user or other devices upon events, even if the user or these devices are located on the other side of the earth. The integration with other devices and their services enables applications that cannot possibly be foreseen and might drastically change how people organize their chores and how they rely on smart devices for everyday tasks.

1.2.3 Home and Building Automation

Home and Building Automation (HBA) is a huge field that was mostly covered by partially proprietary industry standards, and did initially not attract the same advertence as the aforementioned use cases for IoT. It has now become apparent, that considerably more elaborate uses are possible, when the data of single devices are accessible not only for devices in the same system, but also to other devices, and can thus further be used with other devices or services, regardless of their make and kind. In the recent past, more and more HBA devices are brought into the Internet through gateways, Web services or natively, e.g. letting users adjust their climate settings on their commute. Even more interesting, is the undertaking of establishing a full-blown HBA system with smart, Internet-enabled devices, opening up an incredible amount of possibilities to interact with the system and the single devices.

1.2.4 Smart Cities

As cities grow, the corresponding infrastructure is harder to maintain and various parts of this infrastructure require frequent, yet mostly unscheduled, manual intervention due to failures, changes in usage, infrequent public events, accidents, etc. The scale of both, cause and effect of these interventions is so large, that manual changes to the system are mostly too late and misguided. The city of the future, commonly referred to as *Smart City*, is required to handle this better. The solution to this problem begins with *distributed monitoring* of the affected infrastructure and appliances through small and embedded sensors, which enables faster responses to events and improved decision making, e.g. traffic monitoring at major crossroads. Further, a subsequent *automation* of this infrastructure can boost efficiency of such a city on multiple fronts.

The term Smart City also extends to health care improvements and resident satisfaction through faster and more efficient response to their concerns, and is supposed to address various challenges, including the environmental changes of the future. The IoT is already a key element for the success of Smart Cities, as it builds the technological foundation.

1.2.5 Environmental Monitoring

The desire to monitor the environment are as old as humanity itself, because of its impact on our daily life and with the ulterior motive to forecast events that will interfere with our plans. Today, we not only understand the underlying mechanisms better than ever, we also identified additional factors, which we can and do have and influence on, e.g. CO_2 emissions.

Technological advancements enable us to measure a vast amount of values at a multitude of remote locations, without considerable delay. The IoT is a major step forward for these efforts, facilitating data collection and deployment of tiny and mobile sensors, up to real-time wildlife monitoring and sensors floating in the deep sea and in the stratosphere.

1.2.6 Manufacturing and Logistics

Efficiency is a major concern for commercial endeavors, as even small saving per part will yield huge cost reductions with large volumes, thus small investments might result in a considerable

advantage over competitors, or if left out, result in a substantial loss.

In the past, a big effort was put improving the planning of the respective processes. With the advancements accompanied by the IoT, the real-time monitoring of even the smallest parts in those processes is gaining traction, and will improve the efficiency of the remaining processes, which can hardly be scheduled ahead of time.

1.2.7 Energy management and the *Smart Grid*

Since its beginnings, the peak capacity of the electric grids all over the world grew substantially as more and more appliances were connected to it. At the same time, consumers demand a near perfect availability, while the grid operators suffer from highly differential loads during the day, also depending on weather, daytime and season. Additionally, power generation by renewable energies is sometimes harder to schedule, due to the variable availability of e.g. wind and sunshine.

Using the advancements of the IoT for distributed monitoring and control of the affected infrastructure, even down to the consumers appliances, can substantially improve the efficiency and sustainability of the electric grid as we now know it, while the self healing properties of the Smart Grid might at the same time improve its reliability.

1.3 Web Services for the IoT *or* the "Web of Things"

Web services did play a major role for late Internet, enabling the reuse of published Web services for different purposes or facilitating the aggregation of individual services to more elaborate constructs, also called *mashups*. The Web of Things (WoT) draws from the success of Web services and their importance for the Internet at the time being, and attempts to apply their paradigms to the IoT [3], [4]. The WoT is only partially a next step to the IoT, but also a possible way to establish an interoperable IoT in the first place.

Unfortunately, the main difference of the IoT compared to the conventional Internet as we know it, is the inclusion of constrained devices, and Web services and particularly the technologies they are based on, are assumed to be considerably too demanding for this class of devices, primarily regarding energy consumption and memory usage. The research community has yet to decide upon definitive solutions to these challenges before the IoT can benefit from the theoretical possibilities of the WoT.

1.4 Problem Statement

The IoT is commonly anticipated to become an omnipresent reality, and one of the various possible applications is the integration of **constrained devices** (e.g. wireless sensors and actuators for building automation) into such a network and hence make them generally accessible for interaction in-between each other and with them [5].

This future is currently held back by **incompatibility** between these devices, caused by several different, sometimes proprietary, protocols and completely different underlying technology

stacks. A prerequisite for this vision is thus the interconnectivity through a protocol stack based on open standards, based on the tradition of the Internet.

The Internet is currently driven by the IP protocol suite, thus one obvious choice as backbone for the IoT might be an IP-based stack, with a suitable application protocol and a common object representation model on top, which provides interoperability among the heterogeneity of devices of different origin, make and model and even those based on different technologies. **Web services** and the respective paradigms and technologies have proven to facilitate the device-agnostic combination software components through abstraction and well defined interfaces. Bringing the principles of the Web to the IoT, also called the WoT, is thus an upcoming trend and several solutions were already proposed.

Web technologies unfortunately often rely on HTTP/TCP/IP as protocol stack, which makes them **highly unpractical** for constrained devices. A battery powered temperature sensor running such a service would require constant battery replacement, because this stack introduces a considerable demand for processing power and radio transmissions. Even if stripped down to IP itself, such a protocol stack is too demanding to operate on respective devices. On the other hand, the combination of HTTP and TCP lacks some core features essential to automation networks, such as **group communication and asynchronous communication**, which is, under certain scenarios, considerably more efficient, and thus indispensable for constrained devices.

Hypothesis *The suggested solution is to use Web technologies on top of a communication stack optimized for constrained devices which still enables integration of constrained devices into an Internet of Things. Such a solution should be* ***feasible*** *and offer an* ***equivalent feature set*** *to existing solutions, particularly regarding group communication.*

Hypothesis *The possibility for group communication and asynchronous communication allows a solution to* ***scale well*** *for various larger, fragmented and complex network structures and in particular for* ***diverse communication scenarios****.*

1.5 Methods

For the purpose of demonstration a prototype implementation of the outlined protocol stack will be carried out on constrained devices. Following software engineering traditions for embedded systems this implementation will be built against requirements defined by **scenarios and typical use cases**. Overall this should result in a **proof of concept implementation** which is considered sufficient to underline the feasibility of the solution and the practicability of an automation system based on this stack.

The protocol stack will consist of the **Constrained Application Protocol (CoAP)** [6], running on top of **6LoWPAN** [7], a communication protocol to enable IPv6 on wireless constrained devices. For this, Contiki will be used as underlying platform.

On top of this protocol stack, **Open Building Information Exchange (OBIX)** will be used to provide an object model to represent the data points in a standardized manner [8]. Additionally, Efficient XML Interchange (EXI) will be used to reduce the size of the transmitted messages and further to compare solutions using either compressed or uncompressed messages [9].

Further various encodings, parameters and communication patterns will be **simulated** with Cooja, the Contiki network simulator [10], in order to study the resulting real life implications and limitations of the protocol stack especially on a **larger scale** and with different communication patterns, to verify if functional and non-functional requirements can be met.

1.6 Aim and Structure of this Thesis

The aim of this thesis is divided into three goals:

- A Web-based IoT stack will be presented that can be reasonably deployed on embedded and constrained devices.
- A **proof of concept** implementation of the suggested communication stack will be carried out and deployed on constrained devices and thus the feasibility of this solution will be shown. Where possible, its practical benefits and shortcomings will be exposed.
- Regarding the scalability of the presented solution and its aspects, the stack itself and available message encodings and communication styles will be **simulated and compared**. Furthermore the resulting numbers will be presented.

Overall, the results should lead to a better understanding of the practical aspects of the outlined solution and especially its limitations. **Proving feasibility and gathering empirical results** is crucial for the standardization process and a necessity for an agreement on future efforts.

The thesis is structured as follows: In Chapter 2 the current state of the art is discussed, together with the underlying standards used/applied in this work. A brief overview of related work will be given in Chapter 3.

The proposed concept for an IoT stack for constrained devices is described in detail in Chapter 4, and the implementation of it in Chapter 5, listing the used components and summarizing the gathered insights. The subsequent simulations are described in Chapter 6 and accompanied by the respective results.

Finally, a discussion of the solution as a whole will be given in Chapter 7, together with an outlook on possible future work.

CHAPTER 2

State of the Art

2.1 OSI model

During the very early beginnings of the Internet, the International Organization for Standardization (ISO) defined the Open Systems Interconnection (OSI) framework architecture. As part of this framework, the OSI model was defined as an abstract, layered network model, where each layer offered a certain functionality to the layers above, based on the functionality of the layers below (see Fig. 2.1).

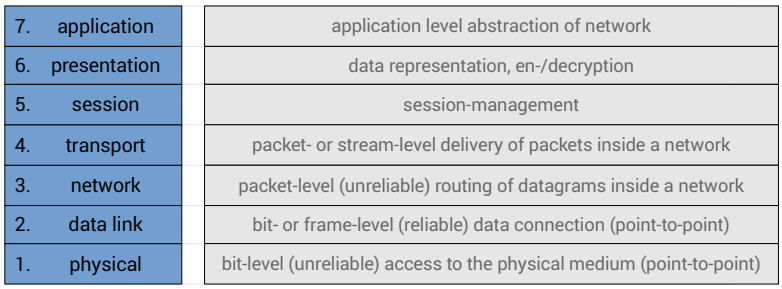


Figure 2.1: OSI model

This layered model (ideally) offers various implementation advantages, e.g. **separation of concern** (every layer is only concerned with its own function) and **flexibility through implementation independence** (layers can be stacked or replaced interchangeably, through abstract interfaces), although this second ideal is only rarely met, as layers for certain technologies may expect specific functionality offered by certain (lower) layers and/or are designed for certain technologies in the first place.

Nevertheless, the OSI model is still the established blueprint for network architectures, separating the functionality of the implemented protocols. As a side effect, the OSI model is capable of showing where interoperability between different network stacks is possible: Whenever a

common layer is implemented on two or more independent network stacks, it is generally possible to achieve interoperability for both stacks through "bridges", i.e. devices that implement both stacks and can forward between both networks.

The standards and protocols presented in the following sections are designed against this model. The resulting stack as a whole will then be presented in Section 4.1.

2.2 Radio Duty Cycling (RDC)

Constrained devices in a Low power and Lossy Letwork (LLN) typically suffer most under one constraint being a reason for all other constraints, which is **restricted energy reserves**, using either batteries or energy harvesting. Further, one of the potentially biggest power drains for constrained wireless node is the operation of the radio, in particular if the device must continuously listen (idly) for incoming traffic. Additionally, most nodes participating in a LLN, due to their constrained nature, encounter relevant traffic only sparsely and a potential data transfer only takes relatively little time. This means that the devices' radio transmitters are mostly idle, which makes continuously listening dispensable.

Under these assumptions, the biggest potential for energy savings is to reduce the time a device is actively listening on the radio to an amount that reflects how seldom transmissions factually are, but without missing sparse transmissions, without resorting to scheduled/slotted transmission intervals.

Radio Duty Cycling (RDC) is based on the idea to duty cycle the radio, which means to only periodically enable the radio to listen for incoming transmissions and again disabling it immediately afterwards (if there was no transmission to receive). To avoid missing all the transmissions that occurred in the meantime, during the deactivated radio, the sender must implement countermeasures, that ensure that the receiver will see the incoming transmission. This is typically done by sending the packet repeatedly until the receiver acknowledges the reception. For non-acknowledged transmissions (e.g. multicast/broadcast), the transmission must be repeated for a certain time-span that correlates with the probing-intervals of the receivers, so that every neighbor will enable its radio at least once during the transmission of the packet and thus a successful reception can be expected. Although this seems to pollute the whole idea, it depends on the parameters of the duty cycle protocol and is essentially a matter of *how often* transmissions are expected to occur compared to how often the receiver can safely disable the radio in return.

Besides the basic principle of Radio Duty Cycling (RDC), a lot of small improvements and refinements can be done, which resulted in different protocols for RDC. The dominantly used protocol in Contiki is ContikiMAC [11], [12], which builds on the various refinements of some of its predecessors:

- Periodic wake-ups by B-MAC [13], X-MAC [14], BoX-MAC [15]
- Phase-lock optimizations by WiseMAC [16]
- Data packet itself as wake-up strobe by BoX-MAC [15]

2.3 IEEE 802.15.4

The IEEE 802.15.4 standard [17] was proposed due to the lack of a standard well suited for Low-Rate Wirless Personal Area Networks (LR-WPANs) for (very local) wireless industrial, residential or medical applications with relaxed demands for data rate and Quality of Service (QoS), but in return with a low impact on complexity, cost and energy consumption, so that it is well suited for wireless connectivity between cheap, self-powered devices [18]. The range of possible applications for IEEE 802.15.4 is long and not restricted by design, so that soon various protocols and specifications based on IEEE 802.15.4 emerged, e.g. *ZigBee*, *WirelessHART* or *ISA100.11a*. For the following chapters, the use of IEEE 802.15.4 as base for 6LoWPAN is of particular interest.

2.4 IPv6

As mentioned in Section 1.1, a huge number of devices is expected to join the Internet in the near future. As of today the number of devices connected to the Internet already exceeds the number of available IPv4-addresses for certain countries or even continents, which can only be mitigated through techniques like Network Address Translation (NAT) or Classless Inter-Domain Routing (CIDR). When IPv4 was defined in [19], the later extent of the Internet was highly underestimated, due to the initial intended uses and targeted audience. Hence, the total number of unique addresses ($\sim 4.3 * 10^9$) was chosen too small, in particular if in the near future the IoT will bring a vast amount of additional devices to the Internet, of which each one is supposed to receive its own IP address.

As soon as the public adoption of the Internet took place, with an always growing number of personal devices being connected, the immanent shortage of IPv4 addresses was clearly foreseeable and hence a successor, IPv6, was defined [20], bringing, amongst other changes, more unique addresses: ($\sim 3.4 * 10^{38}$). The deployment of IPv6 is still underway and although struggling on a broad scale because of being incompatible to the existing IPv4 infrastructure and devices [21], the standard itself and the main infrastructure are ready and well tested.

2.5 6LoWPAN

IPv6 was generally considered to be highly unfit for any infrastructure that is resource-constrained and thus typically bound to low-rate and low-payload communication with little to no overhead involved, e.g. a Wireless Sensor and Actuator Network (WSAN). Hence, tailored standards were used atop IEEE 802.15.4, so that interaction with devices inside of such networks always had to happen over a respective gateway, translating between both worlds.

This perception changed with 6LoWPAN, that defined transmission of IPv6 packets over IEEE 802.15.4 networks through different adaptations [7], of which the most important are:

Packet Fragmentation The minimum packet size of IPv6 is 1280 bytes, which means that every data link layer hosting IPv6 must support a Maximum Transmission Unit (MTU) of *at least* 1280 bytes. IEEE 802.15.4 on the other hand only supports a maximum packet size of 127

bytes, of which in the worst case 46 bytes will be used for the respective headers. This leaves only 81 bytes of payload available to host a potential IPv6 packet, which seems extremely low in respect to the minimum requirements of IPv6. Adhering to [22], 6LoWPAN implements a respective adaptation-layer that implements fragmentation and reassembly of IPv6 packets, as explained in [7].

Header Compression Following the already aforementioned calculation, in the worst case the effective payload left for a potential transport protocol packet inside an IPv6-in-IEEE 802.15.4 packet would be no more than 41 bytes, where the header of an IPv6 packet already has 40 bytes. When considering that the respective transport protocol and a potential application protocol will further reduce this number, as additional headers are necessary, it is tempting to dismiss IPv6 over IEEE 802.15.4 completely, as even the aforementioned packet fragmentation cannot improve these numbers, it will rather worsen them. To make IPv6 packets over IEEE 802.15.4 feasible, 6LoWPAN also defines an IPv6 header compression that is applicable under certain conditions [7], in the best case reducing the former 40 bytes IPv6 header to only **2 bytes**. Further, 6LoWPAN also incorporates compression for the predominant transport protocols UDP, TCP and ICMP, that for example will compress a UDP header (in the best case) from 8 bytes down to 4 bytes, over all already saving 42 bytes compared to the equivalent IPv6-UDP packet without header compression.

Stateless Address Autoconfiguration 6LoWPAN also defines how to form IPv6 interface identifiers from IEEE 802.15.4 interface identifiers [7], depending on if they have an IEEE EUI-64 address (following [23]), or a 16-bit short address, which includes forming a 48-bit address from the 16-bit PAN-ID, the 16-bit short address and 16 zero-bits for padding, then using this 48-bit address adhering to [23].

2.6 Routing Protocol for Low-Power and Lossy Networks (RPL)

Due to the mesh like structure of LLNs, like the ones based on IEEE 802.15.4, energy efficient routing of point-to-point, multipoint-to-point and point-to-multipoint communication in LLNs is non-trivial and cannot be sufficiently solved by methods used in conventional IP networks, which would typically be based on IEEE 802.3 wired Ethernet or IEEE 802.11 WLAN. IETF RFC 6550 [24] defines the Routing Protocol for Low-Power and Lossy Networks (RPL) to solve IP routing for LLNs without restricting itself to certain link-layer protocols, but primarily aiming at 6LoWPAN networks.

RPL organizes a network of devices by building an underlying tree structure, the Directed Acyclic Graph (DAG) (see Fig. 2.2), that is organized around typically one DAG root, which is the root of the tree structure and has no outgoing edge, forming a Destination Oriented Directed Acyclic Graph (DODAG) and the according root becoming the DODAG root. The DODAG root is also eligible as *border-router*, connecting the LLN to other networks which are located outside of the LLN and usually based on different technologies, e.g. IEEE 802.3 wired Ethernet.

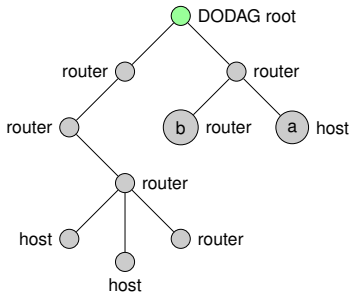


Figure 2.2: Example RPL DAG

Communication is thus separated into *upwards traffic* towards the root and *downwards traffic* (away from the root). Devices taking part in a RPL DAG are either *routers* which forward packets or *hosts* that generate traffic, or both at the same time.

A major feature of RPL is the auto-configuration that leads to the construction of the underlying routing structure and selection of used links, the DODAG, due to the nature of LLNs being unreliable and its topology being subject to change. DODAGs are built and maintained through the exchange of DODAG Information Object (DIO) messages and by applying an objective function for the selection of available links. This results in a distributed algorithm, where link-local multicast DIO messages are being emitted by participating nodes, which leads to the receiving nodes joining a new or maintaining an old DODAG, according to the objective function and the *rank* of their neighbors. The rank of a node is a scalar value which is calculated by the objective function and correlates with its distance to the DODAG root. Every node builds a routing table for the destinations advertised via DIO messages and might additionally save dedicated parents as default routes.

Downward routes depend on the *storing mode* of the RPL instance. If the *non-storing* mode is used, downward traffic is always routed over the DODAG root, from which it is then routed downwards to the destination by IP *source routing*. For the RPL network depicted in Figure 2.2, this would mean that a message from the node labeled *a* to node *b* would be routed via the *DODAG root* node and then back on almost the same route towards node *b*. For storing mode, a message might travel downwards again as soon as it reaches a common ancestor of the source node and the destination node and is routed by the IP *destination address*. In Figure 2.2 this would mean that a message from the node labeled *a* would be routed over the router node adjacent to node *a* and from this node sent directly to node *b*. Destination routes are established by the use of *Destination Advertisement Object (DAO)* messages which nodes send to or via their parents and indicate the destination addresses and prefixes that are reachable via the node that sent the DAO message, i.e. destination addresses and prefixes for which the node has routes. In the case of *storing* mode, the DAO messages are sent directly to the root via the default parents. In the case of non-storing mode, DAO messages are sent to the node's parents, which then calculate the prefixes they have to advertise and then themselves send respective DAO messages to their parents, or omit them, if all prefixes are already advertised.

For its messages, RPL defines a type of ICMPv6 message, the *RPL Control Message*, and also defines an "all-RPL-nodes" multicast prefix **FF02::1A**.

2.7 Trickle

The Trickle algorithm is, similar to RPL (see Section 2.6), designed specifically for use in LLNs and with energy efficiency in mind, but is not at all restricted to this realm [25], [26]. It was initially intended for the distribution of program code updates, but can be used for data dissemination in general.

The purpose of Trickle is the dissemination of information through a network of participating nodes in a polite manner[1]. Figure 2.3 depicts a network of Trickle nodes, where reachable neighbors of a node are indicated by lines between the nodes. The nodes will repeatedly exchange meta-information with each other, describing the information known to each individual node, which is considered a *state* of information that has to be kept consistent. If neighboring nodes agree upon the current state, the interval at which the meta-information is exchanged is exponentially increased up to a maximum value. Once inconsistencies between neighbors are discovered, this indicates a node has a deviating information than at least one of its neighbors, and the node with the newer information will share its updates. Also, the interval for the exchange of meta-information is reset to ensure that the inconsistency is resolved quickly throughout the network. The Trickle algorithm is also designed to be inherently density aware, keeping the overall numbers of transmissions at a constant rate, as each node reduces its transmissions with increasing density. The benefits of such a message suppression is evident, if the network shown in Figure 2.3 is considered. Several nodes are connected to multiple neighbors (high density) and if enough of them already have transmitted the same information, another transmission of this information can in some cases be omitted.

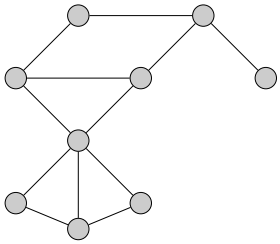


Figure 2.3: Example Trickle network

The Trickle algorithm can be configured through 3 parameters:

- I_{min} the minimum interval for state exchange
- I_{max} the maximum interval for state exchange
- k the redundancy constant

I_{min} and I_{max} set upper and lower boundaries for the interval in which a node will advertise its state. Every interval, a counter c will be reset to 0. Strictly speaking, a node will choose a random point t, that meets $\frac{I}{2} \leq t < I$ where $I_{min} \leq I \leq I_{max}$. At startup, any time or a random time meeting the interval can be chosen. A timer will be set to run out at t which signifies the

[1] Polite in this context means avoiding an unreasonable occupation of the network infrastructure and at the same time conserving energy.

end of the current interval. Should the node receive a state information of another node, which signifies consistency with its own state, the counter c will be increased by 1. As soon as the timer for t runs out, signifying the end of the interval, the node will set I to $min(I * 2, I_{max})$ and transmit its own state information if $c < k$ is still met. Once c reaches k before the timer runs out, the information is considered redundant and the transmission at the end of the interval suppressed. Whenever an inconsistency is detected and $I > I_{min}$, the node will reset I to I_{min} and restart the interval timer with $t = I = I_{min}$. Trickle implementations might also define additional triggers that cause a reset of the interval timers.

An optional delay of a node's transmission by I_{min} when encountering an inconsistency avoids broadcast storms in the network, as all nodes would otherwise immediately transmit their information almost at once. Further, this also avoids evitable energy consumption, as some node's transmissions can then be suppressed by the redundancy constant k, by ensuring that nodes in the direct vicinity had the chance to successfully send their updates first.

One of the known issues of Trickle is that sparsely connected nodes, like those at the outer edge of a network, as seen in Figure 2.3 on the very right, suffer from high transmission counts. Especially for networks with diverse density throughout the network, it is simply not possible to define an optimal redundancy constant k that works equally well for all nodes, considering their differing neighbor counts and the number of redundant transmissions they receive. E.g. in the network depicted in Figure 2.3, the node at the center of the network which has 5 neighbors should optimally replicate new information immediately, even if it received it from all three neighbors to the bottom ($k > 3$), while the nodes with only one or two neighbors will retransmit information, even if was not necessary.

2.8 Multicast routing for LLNs

While UDP multicast routing seems to be generally solved for conventional "full-stack" IPv6 networks, this is not the case for LLNs, in particular 6LoWPAN networks running RPL, where efficient routing algorithms are still discussed and regularly new algorithms are proposed. For Contiki two multicast routing algorithms emerged and can be used for routing of UDP multicast datagrams, Stateless Multicast RPL Forwarding (SMRF) and Multicast Protocol for LLNs (MPL), which will be discussed in the following sections.

Other mechanisms and related work will be presented in Sections 3.4 and 5.1.4.

2.8.1 Stateless Multicast RPL Forwarding (SMRF)

RPL does not inherently solve the routing of IPv6 multicast datagrams, and depends on external routing algorithms for this purpose. One multicast routing algorithm specifically written for RPL is Stateless Multicast RPL Forwarding (SMRF) [27], which relies on the network using the *storing-mode* of RPL and draws from the principle idea of *Parent Flooding* [28].

SMRF uses the information in its routing table, filled by multicast prefix information sent by its children, to determine when a child is interested about a multicast datagram or not, which enables a node to forward a respective datagram only if one of its children is registered for the datagrams multicast destination address in the first place. Additionally, it downright *ignores*

every datagram that does *not* have the nodes preferred parents link layer address as its link layer source address, in other words: packets which are not sent or forwarded to this node directly via its preferred parent. This ensures that each node considers and processes every packet only once, and thus *forwards every packets once at most*, while *no state* about already received or forwarded packets has to be kept. This arguably enables SMRF to be simple and lightweight by design and further avoids error-prone logic.

At the same time, this seemingly elegant omission of any kind of state is also cause of the major drawback of SMRF: **multicast datagrams can only travel the RPL DODAG downwards**, away from the root, as a datagram will never be forwarded if it was not sent by a node's parent. While for some use cases of multicast traffic this is no issue, e.g. if multicast traffic is considered to always originate from the DODAG root, it still might pose a undesirable defect for other use cases. Trying to "fix" this would change SMRF drastically, because this restriction is based on an essential mechanism of SMRF to avoid duplicate packets, which is to ignore datagrams if they are not sent by the preferred parent. As no other comparable mechanism is known to the author, besides keeping track of already received/sent packets as a per packet state, removing this mechanism would consequently abolish the statelessness of SMRF, if duplicate packets are to be avoided. Alternatively, a method to either *selectively* forwarding traffic upwards the DODAG and/or to distinguish between upward and downward traffic would be necessary, which is not guaranteed to solve this problem without compromising the functionality of the mechanism, or degrading it into a broadcasting mechanism.

When forwarding a packet, SMRF uses a short delay D that is either equal to the Channel Check Interval (CCI) of the underlying radio duty cycling algorithm (cf. Section 2.2) or F_{min}, calculated as $D = max(F_{min}, CCI)$, and can optionally even further delay the transmission by parameterizing a constant $Spread$ and calculating the resulting delay as a random multiple of D between D and $D * Spread$, to counteract the effects of hidden terminals. This results in the algorithm being automatically adapted to the applied radio duty cycling algorithm, but also being subject to parameterization. Oikonomou and Philips investigated different configurations in [27], and also compared SMRF to MPL (see Section 2.8.2), unfortunately only for a relatively small and constant payload of 4 bytes and unknown network topologies. Following their findings, SMRF yields smaller delays and reduced energy consumption for higher packet losses [29].

2.8.2 Multicast Protocol for LLNs (MPL)

The Multicast Protocol for LLNs (MPL) uses the Trickle algorithm (see Section 2.7) as forwarding mechanism for IPv6 multicast datagram forwarding [30]. With MPL the forwarding becomes a packet dissemination with **broadcasting characteristics**, which makes it independent from any routing infrastructure as e.g. RPL. This characteristics also is the main advantage of MPL over the aforementioned SMRF, as there is no restriction to which nodes will receive what multicast message (cf. Section 2.8.1) and completely independent of the network topology. On the other side this also means that multicast routing information is not used and instead every participating node is involved, even if it is not necessary, e.g. a node and all its children are *not* listening for a certain multicast destination address, but the packet will be processed and forwarded through this "branch" of the network nonetheless.

MPL encapsulates multicast messages as MPL *Data Messages*, which are sent to their MPL *Domain*, i.e. the entity of nodes listening for the same MPL *Domain Address*. MPL Data Messages contain the *Seed Identifier*, which identifies the source of the message, a sequence number of the MPL Seed, and the multicast message itself.

Additionally, MPL *Control Messages* are used to negotiate the currently known multicast datagrams. Control Messages are a type of ICMPv6 datagram, with a list of MPL *Seed Info* entries as content. The Seed Info contains the minimum sequence numbers for a particular MPL Seed, as well as the sequence numbers of Data Messages that are known. A recipient of a Control Message can thus check if he and the sender of the Control Message have a consistent view of the existing Data Messages.

Two modes of operation are distinguished, which are not mutually exclusive:

- **Proactive Forwarding** Nodes acting as MPL forwarder will transmit MPL data messages immediately, independent of information about their neighbors and their state, using the Trickle algorithm.
- **Reactive Forwarding** Nodes acting als MPL forwarder will send MPL Control Messages (also using the Trickle algorithm) to negotiate the currently known MPL Data Message with their neighbors, and transmit those only if indicated by inconsistency between the participants.

Apart from being essentially a polite **flooding algorithm**, MPL also suffers from a slightly different drawback. Because multicast datagrams are not immediately forwarded but cached, if Data Messages are enqueued in a high frequency, the algorithm has a tendency to deliver datagrams not in the order they were sent, which might be a problem for some applications [27], e.g. Constrained Application Protocol (CoAP) "blockwise transfer" (see Section 2.10.1).

2.9 REST and RESTful Web Services

Representational State Transfer (REST) was defined in [31] as an "architectural style for distributed hypermedia systems", defining a stateless and uniform client/server architecture for software components, abstracting the details of how individual components are implemented, well suited for the structure of the World Wide Web.

The main principle behind REST is the use of *resources* as abstraction for any type of information (in a given *representation*, i.e. the format in which the information is submitted). A certain information is thus accessed via the respective *resource identifier*, which is typically a Uniform Resource Identifier (URI), a substantial component of the World Wide Web itself, overall ensuring a perfect fit of REST in/on top of the World Wide Web.

Following the paradigm stated above, RESTful Web Services describe a uniform interface on a server, that allows for fetching and modification of information via HTTP request methods (e.g. `GET`, `POST`, `PUT`, `DELETE`[2]) on their respective Uniform Resource Locator (URL), using one or more representation formats for the data, negotiable between server and client.

[2]Please note that the definition of HTTP/1.1 specification was influenced by REST, so that some of these methods were added to embrace the ideas of REST [32].

RESTful Web Services are already a well established system design paradigm, for either component integration and even more popular with open interfaces for third parties, e.g. customers or third party developers, because of their uniform design and the underlying well established and ubiquitous technologies, most of all the Web itself. Interactions with RESTful Web Services are also easy to debug against, as the interactions with them are stateless, the received and sent information is typically human-readable and accessible with a vast amount of freely available tools and libraries, in contrast to its proprietary counterparts. Even a basic Web browser can be used to issue a `GET`-request against a resource, by navigating the browser to the respective URL, in result displaying the server's response in the browser window.

2.10 Constrained Application Protocol (CoAP)

The uniform architectural style of REST has proven to be successful as architecture for generic client/server interaction on the Web, and its notion of operations upon resources very well suited for most interactions with generic data. It is thus tempting to look at HTTP/REST when trying to implement Web-accessible resources on a wireless, IPv6 enabled sensor node, but due to the constrained resources typically available, a full-blown REST stack using the HTTP/TCP/IP-suite on a constrained device is a highly unfortunate match. In order to bring RESTful services to constrained devices, the Internet Engineering Task Force (IETF) Constrained RESTful Environments (CoRE) working group was formed and defined, amongst others, the **CoAP** [6].

Being heavily inspired by RESTful Web Services over HTTP, CoAP uses a very similar interface: It also uses verbs for operations upon resources, such as `GET` and `POST`, identifies resources via their URI and supports content-types, as well as status codes for error/success feedback. Additionally, the CoAP standard defines some additional features, which distinguish it from HTTP to further optimize it for its intended use cases:

- CoAP binds solely to UDP as transport protocol, which is well suited for **unreliable, asynchronous, per-datagram, connection-less and energy-conserving communication** in LR-WPANs, and at the same time it relaxes the one-to-one communication constraint which would be introduced with TCP, hence making *multicast-support* for efficient group communication possible.
- It uses a fixed size binary header of only *4 bytes*, optionally followed by additional options, with focus on low overhead and reduced complexity.
- Following message types are defined: *Confirmable*[3]*/Non-confirmable*, *Acknowledgment* (to confirm a Confirmable) and *Reset*,
- CoAP incorporates a standardized *resource discovery*, following [33], so that nodes/servers can be queried for available resources, which fits well with M2M applications.

The close affinity to HTTP makes it possible to quickly translate between CoAP and HTTP at a respective gateway through a "stateless HTTP mapping" [6], facilitating interoperability between both worlds.

[3] *Confirmable* messages are retransmitted until they were either acknowledged or a defined number of retransmissions went by unacknowledged.

Some refinements to CoAP which are not part of the RFC itself are also relevant for this work and shall thus be mentioned in the following sections.

2.10.1 Blockwise Transfer

Seemingly similar to HTTP chunked transfer encoding, the IETF draft *core-block* defines blockwise transfer for CoAP [34], although the reasoning behind HTTP chunked transfer encoding is arguably different, and HTTP mostly relies on TCP/IP fragmentation for the functionality equivalent to CoAP blockwise transfer.

The benefits of fragmentation at the application layer in the realm of constrained devices at which CoAP is aiming, is the capability of the protocol implementation to **limit the sizes of its buffers**, which is very relevant for constrained devices. At the same time it still supports considerably larger content sizes, and ensures compatibility with other protocol implementations with possibly larger buffers, as the actually received chunk sizes are inherently negotiated between sender and recipient.

In difference to fragmentation happening at the network layer, which involves a conversation state, blockwise transfer for CoAP is defined to be completely *stateless*. By the use of header fields for follow-up requests which define the next block to be transferred, every single block can be handled independently, reducing the complexity of the implementation.

Blockwise transfer introduces two options to be used with request and response, *Block1* and *Block2* respectively, although both might be present for control usage in the respective counterpart. By restriction of the field values as powers of two, the block sizes can be kept below *3 bytes* and less if not all fields are defined.

2.10.2 Observe

In constrained environments, *polling* a resource, namely periodically sending requests to determine if its representation has been updated, is far from ideal, considering that it either causes numerous needless requests, or a change goes unnoticed for a dispensable long time. The core principles of REST unfortunately do not cover asynchronous communication that would allow a requester to be repeatedly informed of a resource's changed representation, which would be needed to introduce *pushing* of information, and the concepts used for "kind-of-asynchronous" communication with HTTP, *long polling* and *HTTP streaming*, are too complex to be applied to CoAP.

The IETF draft *core-observe* thus defines the ***Observe*** option for the `GET` request, which indicates the server to update the requester about future resource changes [35], after the `GET` request itself was served. This yields an effective **asynchronous communication** initiated by the observed resource instead of inefficient polling by the requester. The server must thus keep track of current observers and informs all registered observers about updates until they unregister, reject notifications or are deemed unreachable after several notification attempts.

Group Communication

The whole aspect of group communication for CoAP was rigorously discussed and is highly anticipated [36], [37], most prominently, in the *core-groupcomm* draft [38], which aims at defining and standardizing the mechanisms for group communication in CoAP and which has various references to it in the CoAP standard [6].

The draft proposes the use of **IPv6 multicast** addresses and group's URI for the communication inside a group, i.e. a node that intends to send a message to a group sends a respective CoAP message to a multicast address and every member in a group listens on the same IP multicast address for messages that are intended for this group.

The most recent version of the draft defines a CoAP group as "[. . .] a set of CoAP endpoints, where each endpoint is configured to receive CoAP group communication requests that are sent to the group's associated IP multicast address." [38]. While it does not clearly define the term "endpoint", it is assumed to be used in the tradition of service oriented architecture as the service endpoint, i.e. the CoAP resource. The draft further defines the identification of CoAP groups via an IP multicast addresses and port, and mandates the use of the CoAP URI scheme, which also introduces resource paths to a group identifier, which is considered to be inefficient, error-prone and redundant.

The *core-groupcomm* further defines group discovery and also membership configuration, which always comes down to configure a device to listen for the respective datagrams addressed to the group, either pre-deployment or at runtime, where the factual group address might be retrieved by the node via referenced lookup methods. An optional REST interface for group configuration of a device is described as optional interoperability improvement.

2.11 Efficient XML Interchange (EXI)

Extensible Markup Language (XML) is popular as message encoding in the World Wide Web, especially in conjunction with Web Services, as it is both human- and machine-readable. As such it is an obvious choice for all new technologies related to the Web. Unfortunately, if repetitive messages and constrained devices are considered, XML encoded messages are basically large, schematic, declarative text documents, thus XML results in the necessity to (repetitively) parse large documents and further costly transmission of inefficiently coded messages. In the worst case, to gather a single integer value from an XML-encoded message, XML will cause a huge computational and bandwidth overhead compared to binary encoded messages.

Although compression (e.g. zip) would drastically reduce the message size and thus the bandwidth overhead, it would not improve the computational overhead, as the message would still be required to be constructed as XML document at the senders end and be parsed after decompression at the receivers' end as before and *additionally* has to be compressed and decompressed with computationally expensive algorithms.

Efficient XML Interchange (EXI) was defined by the World Wide Web Consortium (W3C) to solve XML compression in a more sustainable approach, which enables (constrained) devices to natively support EXI-encoding and EXI-decoding, in a way superior to XML-parsing, reducing both the computational and the bandwidth overhead, as it can avoid XML altogether [9].

Additionally, conversion between EXI and XML is also possible, thus EXI is potentially human-readable, if a converter is used. EXI messages are streams, consisting of a message header, stating the used options, schema, etc., and a message body, which is a sequence of events that correspond to the elements as they would occur in the corresponding XML document. Events are coded based on a set of grammars, depending on their probability to occur. EXI uses either a *built-in grammar*, if no information about the used XML schema is available, or can be used with a *schema-informed grammar*, which improves processing speed and compression of the underlying XML significantly.

EXI aims specifically at computational efficiency besides compression, making it a viable choice for constrained devices, and offering verbose machine-readable messages just like XML, that can easily be converted into an XML document and vice versa if necessary, but can be used by an EXI-capable device natively without conversion, supposedly making it more efficient as XML itself, as the events are represented in a more efficient way. Thus it was also chosen as common encoding for the OBIX data model besides XML (see Section 2.12), OBIX binary representation and JavaScript Object Notation (JSON) in the respective current draft [39].

2.12 Open Building Information Exchange (OBIX)

Even with a common application protocol two connected devices can only communicate successfully if they agree upon **structure and semantics** of the exchanged messages, conforming to a well defined interface. There are already several solutions to this problem, although the list of available options contains a lot of proprietary standards, others are covered by licensing costs. To add injury to the insult, the world of home and building automation is fragmented, and opting for one known stack and interface model, will often result in being unable to communicate with all other devices and requires the installation of additional gateways, often compromising a good amount of benefits of a particular stack.

To improve this situation, OBIX was defined by the Organization for the Advancement of Structured Information Standards (OASIS) as industry standard, to offer uniform interfaces for generic M2M applications. The OBIX specification [8] lists the following objectives of the standard:

- **XML** Representing M2M information in a standardized XML schema.
- **Networking** Transferring M2M information in XML over the network.
- **Normalization** Standard representations for common M2M features: points, histories, and alarms.
- **Foundation** Providing a common kernel for new standards.

Of major relevance for this work is the fact that OBIX resorts to an object model represented in simple and fixed XML schema, which is not only a good fit for a RESTful architecture, but also human readable (cf. Listing 2.1). The abstractions necessary are defined through *OBIX contracts*. The efforts in regards to *Normalization* and *Foundation* imply that information is well categorized (into data-points, histories or alarms) and that the basic schema of OBIX is easily extensible through the possibility to define additional contracts.

Listing 2.1: Example OBIX Thermostat, taken from [8]

```
<obj href="http://myhome/thermostat">
  <real name="spaceTemp" units="obix:units/fahrenheit" val="67.2"/>
  <real name="setpoint" unit="obix:units/fahrenheit" val="72.0"/>
  <bool name="furnaceOn" val="true"/>
</obj>
```

CHAPTER 3

Related Work

3.1 The "Web of Things"

Deployment of Web services, respectively their paradigms, to constrained devices, in order to achieve a truly interoperable IoT has been an ongoing topic in the research community. Several proposals to the problem were described so far, and while the motivation and intentions behind them are vastly congruent with each other and this work, their approaches may differ vastly.

Some solutions skip the definition of network or transport layer protocols for the used protocol stack, as the one described in [40], where eXtensible Messaging and Presence Protocol (XMPP) extensions for the use in a WSAN are proposed, using XML messages similar to OBIX (cf. Section 2.12). The protocol itself was completely implemented at the MAC layer, circumventing IP/IPv6 and any potential transport protocol.

In the tradition of the Web Service Description Language (WSDL) and SOAP Web Services (WS), the Devices Profile for Web Services (DPWS) specification was formalized as standard by OASIS [41]. Moritz et al. acquainted their implementation *uDPWS* for IP-enabled WSANs using Simple Object Access Protocol (SOAP) messaging in [42], together with a *SOAP-over-CoAP* binding. As the proposed solution using SOAP messages results in comparably large messages, they also introduced *uEXI*, an EXI parser implementation for constrained devices. While the goals and intentions of the presented work are largely congruent with the ones of this work, the author of this work does not consider the SOAP/DPWS stack a competitive alternative to OBIX because of the message sizes and/or the large and complex XML schema it results in.

SensorsMW is a middleware in the tradition of Service-Oriented Architecture (SOA) focused on QoS and management of WSANs, using the *WS-Agreement* framework [43]. The Web services are seemingly provided by a gateway, which acts as a controller for the WSAN, and as such caches information, and translates requests to commands understood by the constrained devices in the WSAN. The authors undertook an implementation based on TinyOS on Crossbow MicaZ motes (ATMEL ATmega128), with an unconstrained device as gateway, arranged in a simple star topology. Unfortunately, no evaluation results were presented. Further, the indirection over the required gateway is considered to be a single point of failure and introduces avoidable com-

plexity to the system and implies that the constrained devices are not directly accessible and most of all cannot access each other over these interfaces.

Souto et al. present *Mires* as message-oriented middleware with focus on publish/subscribe functionality in [44], [45] for an unattributed operating system.

Madden et al. propose the content-based *TinyDB* for TinyOS [46]. TinyDB comes with the *Acquisitional Query Language*, drawing on SQL, were queries of the type "SELECT-FROM-WHERE" are used to issue data collection from sensors and supports several advanced constructs, e.g. "aggregation queries" or lifetimes for queries. To efficiently forward queries through a network (or avoid dissemination if not necessary), TinyDB defines *Semantic Routing Trees*. Apparently, TinyDB thus installs a route-over infrastructure in a WSAN, but it is unclear to the author, if queries are sent through the network "as is" or if they are encapsulated in or translated to a message structure more suitable for constrained devices. This also applies to the query results being sent back by the sensors to the TinyDB root. While TinyDB does solve how sensor data can be retrieved, it does not inherently solve how clients can connect (directly) to sensors, besides sending a respective query to the TinyDB root.

Another protocol focused at publish/subscribe functionality as extension to the Message Queuing Telemetry Transport (MQTT) protocol (designed for telemetry applications) is *MQTT-S* optimized for WSANs [47]. The paradigm is not implemented solely between publisher and subscriber, but handled by a broker that receives both subscriptions to topics and the according published data. This broker is a potential single point of failure. Further, it offers potentially inefficient communication patterns, as every datagram has to pass the broker outside of the WSAN, making efficient group communication infeasible. A proof of concept implementation was done on various ZigBee and TinyOS (MicaZ) motes.

VITRO is another abstraction middleware approach, that uses an unconstrained service gateway outside of the WSAN to offer services of the motes inside the WSAN to clients, calling the concept *Virtual Sensor Network* [48]. The main advantage of the *VITRO Core Framework* is the potential service abstraction of various proprietary and else incompatible sensor networks through legacy gateway interfaces to a uniform service front-end, but again poses as potential risk as single point of failure. At the same time, the architecture of a sensor node stack is presented, that realizes "native" virtual sensor nodes at the sensor device level. The architecture seems unnecessarily complex, and no implementation details are conveyed. Further, the benefit of the architecture is unclear and the feasibility for constrained devices not demonstrated, as the work focuses on the architectural details only.

Linksmart, proposed in [49] tackles the idea of semantic Web services for embedded devices but only offers a description of the presented stack without any further details for the implementation on constrained devices. From the amount of plotted features of the stack, an implementation on smaller constrained devices (e.g. TI MSP430) seems improbable.

Hughes et al. introduce *LooCI*, a component-based approach to WSAN middleware for *JavaME* based sensor nodes in [50]. Its primary goal is to facilitate device reconfiguration through the "LooCI Reconfiguration Engine" which is also responsible for the wiring of components on the individual sensor nodes in the first place. The messaging between the components follows an event-based paradigm that offers publish-subscribe functionality between individual nodes and is based on Remote Procedure Call (RPC) interaction. *LooCI* implements own net-

working components but does not mention any specific insights on optimizations for constrained wireless sensor nodes. Further, the author of this work does not consider *JavaME* a competitive foundation for constrained devices.

In the course of the *BuildWise* project, Menzel et al. present a prototype of a "hybrid" network that connects wireless sensors into existing, established, wired HBA systems [51] (cf. [52]). The prototype is implemented on Tyndall motes, using a modified IEEE 802.15.4/ZigBee stack and *LON Standard Network Variable Types (SNVTs)* for data management and communication.

The *SWE* initiative by the Open Geospatial Consortium, incorporates many established and open standards, e.g. Sensor Model Language (SensorML), Transducer Markup Language (TML) and Sensor Observation Service (SOS) to build a framework to access various types of sensors [53], primarily intended for geospatial sensor readings. The framework makes use of the transducer interface standard *IEEE 1451* to avoid development of independent transducers for every possible sensor network. It aims at unifying sensor readings for sensor devices of various kinds, and simplifying data acquisition, which is distributed over several components and services.

Triantafyllidis et al. present a system architecture for health monitoring applications, inspired by *SWE*, also using SensorML, to ease adding new sensors to the network [54]. The proposed solution works allegedly without centralized logic in order to reduce traffic to and from constrained devices and exploiting available computational power in more recent devices, shifting logic to the sensors. For communication with the sensor devices from outside the sensor network, a *Mobile Base Unit* acts as gateway and facilitates data (de-)serialization. Similar to *SWE*, every sensor is registered with this device and their data is aggregated. Sensors can be configured at the *Mobile Base Unit*, from where their configuration is sent to them via SensorML messages, so that they only send data/events if e.g. a certain threshold is reached. A proof of concept implementation was done in *JavaME* (Sun SPOT) and using Bluetooth as radio link between *Mobile Base Unit* and sensor devices. This implementation does first of all not support any links between the sensor devices, thus sensor-to-sensor or sensor-to-actuator communication is only possible over the *Mobile Base Unit*, and further is expected to be infeasible for the device class this work is aiming at.

GSN is a middleware which aims at simplified application development with WSANs through the use of *virtual sensors*, posing as abstraction element for data from real sensors or other virtual sensors [55]. The virtual sensors are hosted by GSN nodes, which are in turn connected (logically and physically) to the physical sensors and map them to the virtual sensors based on XML configuration files. Although the approach offers the possibility to interconnect heterogeneous networks and simplify development of applications based on sensor data, the GSN nodes pose as indirection and as a single point of failure, and a way to communicate directly with sensor nodes, or enable node-to-node communication between heterogeneous devices is preferable.

Aiming to ease energy management in buildings, *WSN-BMDS* is introduced by Spinar et al. in [56]. Gathering data from the sensor motes is not done by communicating directly with them, instead it makes use of *GSN* to aggregate data and publish the respective services. The data itself comes from IPv6 enabled sensor motes (Tyndall and TMote Sky running a b6loWPAN stack), connected to the GSN through b6loWPAN border routers.

The architecture presented for the *HOBNET* project aimed at connecting heterogeneous building intelligence systems follows an approach where legacy systems are accessed via a *Building proxy*, and 6LoWPAN devices can be accessed directly over standardized protocols, e.g. IP and UDP [57]. The proposed approach uses a new XML document schema *Compact-DataRecord (CDR)* for data transfer to and between sensors. According to the architecture description, the application logic is intended to be implemented strictly *outside* of the WSANs, instead of interconnecting the individual devices.

ICSI WSN is yet another middleware that uses virtualization for logic sensor/actuator components to reduce development complexity but in difference to the aforementioned Virtual Sensor Network approach, does not rely on external components. Somewhat similar to the stack presented in Section 4.1, the framework uses an event-driven messaging scheme on top of CoAP with messages encoded in EXI/XML, mentioning 6LoWPAN and IEEE 802.15.4 as underlying technologies and IPv6 addressing in the examples. The structure of the messages exchanged between the nodes is not presented, and the communication pattern between individual components is restricted to the publish/subscribe paradigm [58].

SMAP, introduced by Dawson-Haggerty et al. in [59], offers RESTful Web services on wireless sensors, using HTTP and JSON encoding for data transfer, while the used data representation is drawing from SensorML. For push semantics for resources, the sMAP framework adds a *reporting* data point for a resource, where clients can add themselves as listeners (cf. Section 2.10.2). The complexity of this stack (HTTP/JSON) is hidden for constrained devices through an simplified version, that is converted to and from at the border router to a constrained network, making use of EBHTTP instead of HTTP, UDP instead of TCP and *binary* JSON instead of JSON. A proof of concept implementation was done on TinyOS motes offering data from AC plug load meters and an exemplary message size of a request in the compacted stack (using binary JSON and EBHTTP) compared to the respective request of the full-blown stack. It was also shown, that with sMAP syndication of the data to external data sinks is feasible through data model adapters. The IETF draft for EBHTTP though expired without a successor, and EBHTTP has been credited as inferior to CoAP in the literature due to substantial flaws [60].

In order to create WS-*-style mashups, Guinard et al. implemented RESTful HTTP resources on embedded devices, with *Smart Gateways* for constrained devices not capable of providing the IP/HTTP stack, providing virtual devices that map to proprietary protocols [4], [61], [62]. For a proof of concept, proprietary bluetooth devices were made accessible as RESTful resources via a gateway translating the JSON encoded requests to the proprietary protocol and vice versa, and the full-blown HTTP stack was deployed on Sun SPOT motes (written in Java ME), although a proxy gateway was still necessary to translate HTTP/IP requests for the motes, as they would at the time not support IP. The missing push semantics of HTTP is solved by feed subscriptions, where the node would receive a rule that triggers resource updates and an address of an Atom(Pub) server to push the updates to using AtomPub, although this implies that every listener must implement an Atom server, which is disadvantageous for constrained devices.

3.2 Message Encoding for the IoT

Traditional Web services are in this respect ideologically tightly coupled to XML as predominant message encoding language. Several higher level formats/schemata using XML emerged, e.g. SOAP, to facilitate Web service interoperability and enabling mash-ups of Web services. This led to several approaches that use XML and further also SOAP for resources on WSANs, although due to the limited resources on constrained devices and due to the lack of support for Web technologies IP/TCP/HTTP, this often led to the use of gateway devices, which map virtual devices that are accessible through conventional Web services with the real devices through proprietary, compact protocols and device representations (e.g. ZigBee, Bluetooth, KNX) [43], [53], [55]

As the goal of a truly interoperable and interconnected IoT requires native access to devices without gateways or brokers, with homogeneous interfaces also for constrained devices, and full-blown SOAP and XML are often not feasible for these constrained devices, recent publications often mention binary encodings for messages in Web services for WSANs. For constrained devices, not only the size of the messages itself is important, but also the efficiency of encoding and decoding, or preferably efficient serialization and de-serialization.

Helander introduces the use of SOAP for embedded systems communication, under the assumption that 32-bit microcontrollers are (or will be) cheap and efficient enough to replace proprietary, device specific protocols on embedded devices with the more complex and heavier protocols as SOAP and WS, that are predominantly used on the Internet for Web service interoperability [63]. Some thoughts on XML compression are shared, e.g. *directly generating an encoded event stream* from message templates without constructing an intermediate XML document in the first place (cf. Section 2.11), further the author suggests that devices not capable of holding the full stack could use this message format as highly efficient (essentially) raw binary data with only the smallest necessary subset of predefined encoded events, while more potent devices can convert it to human-readable XML messages. The results for the binary footprint of the implemented stack substantially exceeds the goals set for this work aiming at much smaller, self-powered devices.

As already mentioned in the previous section, Moritz et al. proposed a *SOAP-over-CoAP* binding in [64], applying DPWS in the tradition of WS (adhering to the Web Services for Devices (WS4D) initiative) and further evaluated the resulting message sizes, before and after EXI encoding. The stack at question was extended and optimized in [42], with *uDPWS* as DPWS implementation and *uEXI* as EXI parser, both specifically aimed at constrained devices. The stack was deployed on *TelosB* motes to show the feasibility. The presented results for the binary footprint of *uEXI* are promising. The message sizes resulting from EXI encoding were substantially improved and are at least competitive when the "extended schema" and bit aligned encoding was used. Nevertheless, the binary footprint of the extended schema on real devices is quite substantial due to the large XML namespace, and the feasibility in a real environment remains questionable. The authors mention that for every application a trade-off between the implemented schema and the resulting message sizes must be found.

In [58], ICSI WSN a middleware for RESTful Web services for constrained devices is described, using EXI encoded XML messages over CoAP for event messaging, unfortunately with-

out particular results or a comparing evaluation regarding EXI.

Castellani et al. proposed the CoAP implementation *COAPP* on TinyOS, using EXI compressed XML documents, together with *libEXI*, an EXI processor for constrained devices [65]. Additionally, the efficiency of EXI compression was compared to various other compression techniques, where EXI proved to be outright superior, especially if the *bit-aligned, schema-informed* mode is used. On the other hand, they came to the conclusion that *bit-aligned EXI compression is not practicable for constrained devices*. Unfortunately, the chosen XML documents and the used schema were not presented.

In [66], Caputo et al. introduced a modified version of EXIP [67], that does not use more than 8KB of RAM. This unfortunately led to the *removal of schema informed encoding* and *decoding in general*. The library was then used to encode XML messages on a STMicroeletronics STM32W108 development board, running a CoAP server atop Contiki, although the used CoAP engine was not mentioned. The development board used is significantly more potent than a typical constrained wireless sensor mote regarding CPU clock speed, but with slightly better than average ROM and typical RAM specifications. Comparisons for the resulting message size between the XML-message and the EXI-encoded counterpart were presented, showing a significant advantage for EXI. Numbers for power consumption of the EXI-encoding were not presented.

A method for EXI processor code generation from an XML schema is proposed in [68], which promises a *significantly reduced binary footprint* compared constructing an intermediate XML document firsthand and using an EXI parser that converts the XML formatted message to EXI. Additionally, the generated code is limited to the used schema only and can only produce events from the underlying schema (*STRICT mode*), thus avoiding generic functionality further reducing code size. The produced code supports bit-aligned, schema informed EXI, without any repercussions on the binary footprint, compared to other approaches. The code generation framework is further presented for *Java* code generation, and running on a network of ZigBee motes (TI MSP430), which provide a *Java ME* platform, encoding SOAP messages between sensors and actuators in a small building automation setup. The reported resulting EXI message sizes show that the use of SOAP messages can be practicable for constrained devices, depending on the size of the used schema *per node*. This is also the limiting factor of this approach: With various diverse applications in one network, all together contributing to an extensive collaborative schema (in particular if SOAP is used), it might be necessary to limit the schema to subsets supported by the individual nodes. The binary footprint result is overall comparable to the one produced by *uEXI* in [42], despite the presumably superior design of the library, which is probably attributed to the execution in *Java*.

In [5], the efficiency of EXI, *Fast Infoset*, *BXML*, and XML encoding is compared concerning the resulting message size for each encoding, with results significantly in favor of EXI. It also mentions the *binary representation* of of OBIX or American National Standards Institute (ANSI) C12 as viable choice, although it was not considered for the comparison, which unfortunately are not inherently human readable without translation to XML. As another very popular alternative to XML used in conjunction with RESTful Web services, namely JSON encoding is also mentioned, but also not considered for the comparison.

A thorough comparison of various encodings, plain XML, EXI and *Protocol Buffers* [69],

for messages in HBA environments was done in [57]. While schema-informed EXI encoding generated slightly better results than Protocol Buffers (6% and 9% of the original message size), schema-less encoding is far off (27% of the original message size), yet arguably still better than the original XML message. The paper states that Protocol Buffers has less complexity and results in reduced firmware sizes, without going into detail about this assumption, and under the supposition that XML to EXI conversion must take place to construct EXI messages.

Gligoric et al. undertook a performance evaluation of XML in terms of power usage, memory consumption and complexity in [70] on an Android powered device and compared it to Protocol Buffers with a significantly better outcome for Protocol Buffers regarding all measured metrics. Unfortunately, the results are only partially applicable for WSANs, as the feasibility of a Protocol Buffers implementation on a constrained device is not shown, and Protocol Buffers messages lack the possibility of low-complexity translation to an XML equivalent and thus the potential human readability of EXI encoded messages.

3.3 Group Communication for WSANs

Constrained devices in WSANs are rarely set up isolated, but often fulfill a certain purpose as a group, e.g. multiple temperature sensors and the respective actuators in one room. Especially if HBA systems are to be implemented using WSANs, efficient group communication is a requirement.

As CoAP is arguably mainly targeted at WSANs and the like, the apparent need for (efficient) group communication facilities for WSANs led to the definition of a group communication context for CoAP as IETF draft described in Section 2.10.2.

A different approach is proposed by Ishaq et al. in [36], which masks the individual interactions with single resources belonging to a group, called *entity*, with a single interaction with an *EntityManager*, a proxy object residing as endpoint on any device in the network. This solution is particularly helpful, if the members of a group are queried for a certain value, and the final result is supposed to be an **aggregate value** of all single values. Although this approach partially avoids interactions with all single participants of a group and the resulting complexity, the node hosting the EntityManager poses a single point of failure and individual messages still have to be sent from this device to all group members. Under certain network topologies, and in particular with changing topologies in mind, this proxy behavior might cause adverse effects to the network compared to other proposals.

In [37], Jung and Kastner argue for an efficient group communication that offers similar semantics to KNX groups, but based on top of the proposed IoTSyS stack. They use **IPv6 multicast addresses** as group identifiers, to which not endpoints, but data points can join, as long as they are semantically compatible with/transformable to the groups data-type and vice versa. This makes it possible for an endpoint to have different data points joined with different groups, and obviates sending of a resource identification e.g. a URI with every request. This work draws from the proposed group communication approach, as will be described in Section 4.2.

3.4 On Multicasting in LLNs

Conventional flooding of a WSAN by using broadcasting mechanisms for multicast messages might introduce dispensable messages to uninvolved parts of the network and thus cause increased energy consumption. On the other side, construction and maintenance of a more advanced multicast routing infrastructure is costly and can quickly exhaust the available resources on constrained devices typically used for LLNs. Algorithms used in conventional IPv6 networks are mostly unviable for constrained environments, and several more advanced ideas on the matter of efficient multicast routing in WSANs were proposed.

A comparison of unicasting to various multicasting engines for group communication based on Contiki was already done in [71] and will be extended and discussed with more detail with this work.

A thorough comparison of different broadcast algorithms in RPL was done in [28], and while these algorithms are not intended to solve multicast routing, the described findings and ideas are very relevant for multicasting in WSANs. For their analysis, the authors defined the constraint that traffic will always originate from the DODAG root.

Silva et al. investigate into the suitability of multicast algorithms aimed at traditional IP networks for WSANs [72], through analysis and simulation using a network of Contiki nodes and a modified *uIP* stack and the *NS-2* simulator. The compared algorithms were Multicast Ad Hoc On-Demand Distance Vector (MAODV), *PIM-SSM* and *AODV*. While the suitability of the mentioned algorithms is more or less out of the question nowadays, and the Contiki network stack has considerably changed (with IPv6 support through 6LoWPAN) the stated reasons for the importance of multicasting in WSANs and the results in favor of multicasting over broadcasting remains. Also, the paper gives an early insight in the importance of IP connectivity (and IPv6 compatibility in particular) in constrained environments. The question of which algorithms are suitable instead has yet to be answered.

RFC6550 defines "Multicast operation" for RPL [24], which describes how group registrations should be handled. In particular, the use of unicast DAOs is advised for multicast group registrations, and it is proposed, that the same DODAG structure that routes unicast traffic can be used to route multicast traffic, as long as the *Storing* mode of operation is supported by the routing nodes. Each RPL router would hold multicast routing states, that lists all multicast groups it would form a route for (for which it previously received a group membership registration that passed by it on the way to the DODAG root). Although the standard explicitly does *not* cover "[...] a full description of multicast within an LLN" [24], it suggests that each router would pass an encountered multicast packet to its preferred parents (or to the alternatives if that fails), and to all registered children of that router (but not the origin of the datagram).

The flow of multicast datagrams suggested by [24] is difficult to implement, as Oiknomou et al. point out in [27], as it would require individual unicast frames to be used for every potential child, as multicast datagrams cannot be selectively sent to a subset of neighbors. To improve upon this, Oikonomou et al. proposed *SMRF*, which is discussed in Section 2.8.1.

The authors of SMRF also did a thorough evaluation of the routing algorithm and compared it to MPL in [27], [29], with the conclusion that MPL achieves a better delivery ratio than SMRF at the cost of increased latency and higher energy consumption. This is mainly due to the fact

that SMRF deliberately restricts datagrams to be sent only once per router, where MPL will by design ensure that every node receives every new datagram eventually (cf. Section 2.8.1).

In the literature the reliability of both, SMRF and MPL are often criticized as leaving a lot to be desired [73]–[75].

Tharatipayakul et al. propose *iACK* as improved iteration of SMRF in [73], which uses *implicit acknowledgments* to improve its reliability. Implicit acknowledgments, in contrast to *explicit* acknowledgments, exploit the fact that forwarded packets in wireless medium are generally visible to all the neighboring nodes, including the *previous* sender and thus can be used as implicit reception acknowledgment. *iACK* uses a retransmit queue and a list of its DODAG children, to determine if a packet has to be retransmitted or can be discarded because all children have already forwarded the packet themselves. It is unclear to the author how *iACK* handles DODAG children that are registered for the same multicast address, thus interested in a multicast datagram with a certain multicast destination address but don't have any children themselves registered for this multicast address, thus suppress the retransmission following the basic algorithm of SMRF. This constellation would lead to expected but missing implicit acknowledgments.

As the name suggests, Sensor Node Overlay Multicast (SNOMC) proposed by Wagenknecht et al. in [74] does not use IP multicasting, but *overlay multicasting*, where the multicast routing is done in the application layer instead, and is built with code updates as main use case and in competition to code dissemination algorithms (e.g. Trickle). SNOMC builds a multicast distribution tree and uses *negative acknowledgments* and caching to ensure reliability. Its performance is thoroughly evaluated in [75]. It is expected that application layer protocols like CoAP will need special adaptations to use overlay multicasting algorithms that operate in the application layer themselves, which is seen as disadvantage compared to traditional network layer multicasting, also because it cannot easily take advantage of the routing information that might be available at the network layer, with e.g. RPL.

Other multicast routing algorithms aimed at WSANs are (without specific order) *Directed Diffusion* [76], *Multipoint Relay* [77], *PIM-WSAN* [78], *uCast* [79], *PSFQ* [80], *TinyCubus* [81], *Low-Power Wireless Bus* [82], *Splash* [83], and the routing algorithms described in [84], [85].

Also worth mentioning are multicast routing algorithms proposed for other technology stacks, which most certainly are not fully applicable for RPL networks based on 6LoWPAN, but some of their results might be worth considering, e.g. *Z-Cast* for ZigBee cluster-tree routing [86], or an improved iteration of Z-Cast, presented in [87].

3.5 WSAN Simulations

In [88], network layer simulations of ZigBee are executed using OMNeT++. For this purpose, the ZigBee network layer had to be implemented in OMNeT++ and albeit the scope is different this work's scope, it might be taken as reference for a detailed analysis of the network behavior of the proposed stack. This approach was not followed for this work, as not only a 6LoWPAN implementation for OMNeT++ was missing, but also an application layer implementation for CoAP and a way of modeling a message flow would have been required, and the feasibility of this approach in OMNeT++ was unclear.

Closely related, in [89] a network layer implementation of 6LoWPAN for OMNeT++ is presented, by extending Contiki with a new platform *omnetpp* and using the resulting platform firmware in OMNeT++. The results were unfortunately not available when this work was commenced, but it is a promising foundation for future work using OMNeT++ and with more focus on the network layer of the proposed stack.

CHAPTER 4

Concept

4.1 An IoT Stack for Constrained Devices

Figure 4.1 depicts the proposed stack based on 6LoWPAN and CoAP alongside the OSI model and an alternative stack for full-blown, conventional IPv6 enabled devices.

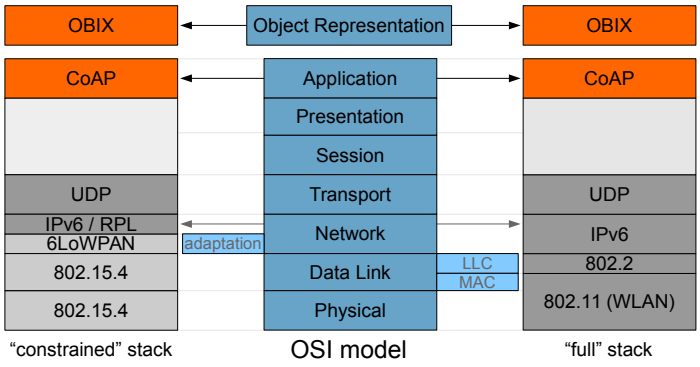


Figure 4.1: CoAP stack

Most parts of the CoAP stack for constrained devices are described in Sections 2.3, 2.5, 2.6, 2.10 and 2.12). Please note, that CoAP is explicitly *not* restricted to be run atop IPv6 as network layer, but for constrained devices it makes little sense to implement two independent network layer protocols at the same time, and thus the constrained stack suggests that possible counterparts also support an IPv6 capable network stack. Otherwise, a bridge might be used to translate e.g. between Fully Qualified Domain Names (FQDNs) and IPv6 addresses. For most current CoAP implementations, IPv6 is a de-facto requirement, and if IPv6 is considered as essential enabler and core component of an interconnected IoT, it is only fair to assume it as least common denominator. Thus, with the use of a corresponding router the stacks are interoperable from the network layer upwards.

A schematic of the IoT using the proposed protocol stack is shown in Figure 4.2.

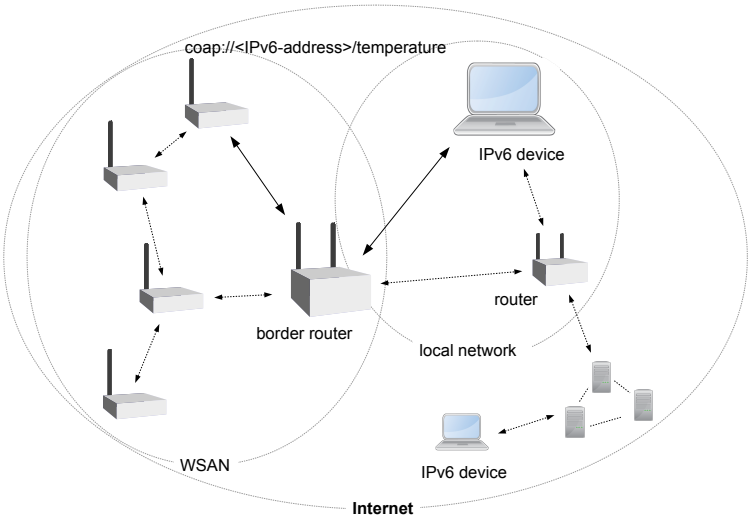


Figure 4.2: Schematic IoT setup

The 6LoWPAN devices residing inside the WSAN are physically connected to the local network via the border router that has the interfaces to communicate on both links, IEEE 802.15.4 on one side and Ethernet (802.3) and/or WLAN 802.11 on the other. This is necessary, as conventional Internet devices are not capable of receiving or transmitting IEEE 802.15.4 packets, very similar to the necessity of wireless access points to bridge wired Ethernet and wireless WLAN connections.

As 6LoWPAN is used as IPv6 adaptation layer, the devices inside the WSAN can be addressed and identified by their IPv6 address. Through the use of CoAP, other IPv6/CoAP devices on the Internet can further access the wireless sensors or actuators following the REST paradigm. E.g. the IPv6 device in the local network in Figure 4.2 sends a CoAP `GET` request to the server `coap://<ipv6-address>` on the resource `temperature` (resulting in the URL `coap://<ipv6-address>/temperature`), to which the corresponding server will answer with the current value of the temperature. A single device can further host multiple resources at the same time, identified by their distinct URL, which further can be nested, e.g. `coap://aaaa::1/thermostat/set-point/`.

Unfortunately, a common application protocol does not guarantee that devices can communicate. Just like people need to talk the same language to understand each other, CoAP devices need a common semantic encoding of information to decipher the information sent with requests and responses. The OBIX standard presented in Section 2.12 was chosen to solve this problem for the proposed stack. An example for a response by a respective object is shown in Listing 4.1.

Listing 4.1: OBIX Example Object, taken from [8]

```
<obj href="obix:obj" null="false" writable="false" status="ok" />
```

Additionally to the described manually initiated interaction with wireless sensors and actua-

tors, CoAP devices can also autonomously interact with each other, e.g. querying other device's sensor values, calculating control logic, and sending commands to actuators on yet another device.

4.2 Group Communication

As already mentioned in Section 1.4, efficient group communication is a necessary feature of the proposed protocol stack, if it is intended to be a viable alternative for existing HBA systems. In the proposed stack, group messages do not differ from regular messages regarding their structure or content, but are forwarded in the network to a functional group of nodes, e.g. several light actuators which are controlled by a single light switch.

A schematic of how functional groups in a WSAN could look like is shown in Figure 4.3, where a functional group of four nodes is identified as group A.

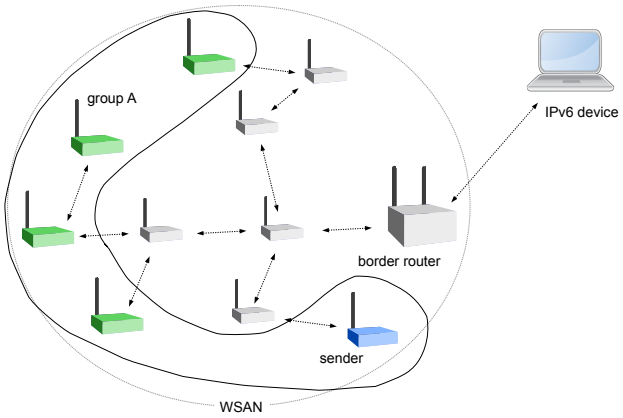


Figure 4.3: Schematic of group communication in the IoT setup

Messages sent to this group shall be received by all members of this group, no matter where the message originate from and regardless of the group members arrangement, i.e. if they are actually in each others vicinity. Note that the upper most group member in Figure 4.3 is only reachable via several intermediate nodes, and that a potential client sending a group message identified as sender is located at the other end of the WSAN, which should still reliably work. Also group messages originating from outside of the WSAN should be possible, e.g. coming from the device identified as IPv6 device in Figure 4.3.

In the opinion of the author, there is still a major issue with the current version of the CoAP *core-groupcomm* draft [38], regarding the relation of nodes and multicast addresses. The draft *seems* to consider additional URI assignments to a group, additionally to the IP multicast address itself. Additionally, it only considers the respective *nodes* participating as group members, instead of their *resources*, which makes it impractical for group communication in the tradition of established HBA systems, e.g. KNX, where **data points**, also called *shared variables*, are part of a group, not their hosting devices.

The proposed stack does instead consider individual CoAP *resources* as members of a functional group, resulting in the setup depicted in Figure 4.4, where multiple resources, identified in the figure as `light`, belong to either group *A* or group *B*.

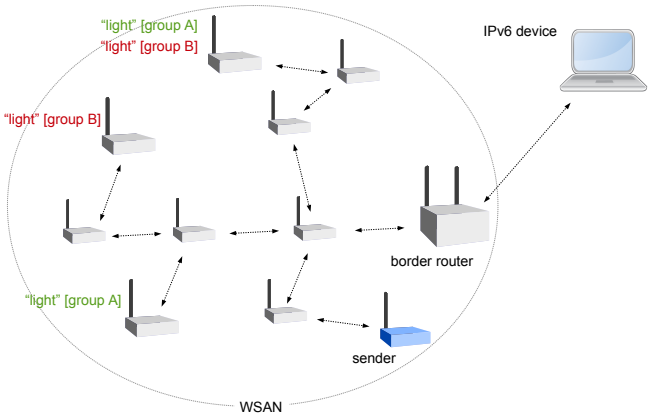


Figure 4.4: Group Communication in the proposed IoT setup

Instead of nodes taking part as members in a group, individual resource endpoints on the nodes are assigned to groups, so that these resources can then be handled as a functional group. In the scenario depicted in Figure 4.4, a sender of a group message could address all light actuators belonging to one group, and thus enable all the lights in the respective room at once. This approach makes room for nodes having multiple resources taking part in different groups, and further the same resource being part in multiple groups at the same time, which serves considerably more use cases.

One detail not explicitly covered in this work is how the **compatibility of heterogeneous resources** is ensured in regard to the group message structure. While *identical* resources are trivial to be addressed as a group, which means they are handled as if they were a single resource and the messages just duplicated for all other members, resources that are *distinctly different* in structure but should still be *functionally* grouped require a set of rules to ensure compatibility. A possible reoccurring scenario is when one button press on a remote will switch on and configure the A/V devices, dim the respective lights, lower the screen and close the blinds by sending a single message to the corresponding group (cf. [37]).

4.2.1 Multicasting for Group Communication with CoAP

One obvious solution to sending UDP datagrams to multiple devices in an IPv6 enabled network is the use of IPv6 multicast datagrams, which is also the approach suggested by the IETF draft for group communication with CoAP mentioned in Section 2.10.2.

Groups are identified by their multicast address, e.g. `FF15::1`, to which messages can be sent and will thus be received by all group members listening for datagrams on this multicast address. Hence, CoAP resources join a group by registering them with a multicast address associated with

the respective group. The routing of multicast messages and registration of multicast listeners with the adjacent devices in the RPL network are then handled by the underlying multicast engine. Please note that this is slightly different from the approach of the *core-groupcomm* draft (cf. Section 2.10.2).

As mentioned in Section 2.8, multicast routing in LLNs can not yet be considered as solved, in particular the scenario shown in Figure 4.3, with the sender of the multicast datagram located on a distinct branch of the WSAN. With the SMRF multicast routing engine described in Section 2.8.1, this datagram would never be delivered (cf. Section 2.6).

SMRF and MPL were initially not available when the work on this concept was commenced. To temporarily improve upon this, and to offer a later alternative to SMRF and MPL a simple **flooding** mechanism was implemented, based on the **LOWPAN_BC0** broadcast header of 6LoWPAN. In difference to multicast routing, flooding will deliver a datagram throughout the network and might cause dispensable transmissions, if only a small group of nodes was intended as recipient, but on the other hand, it has a low complexity and a low overhead. The LOWPAN_BC0 header contains a sequence number, which can be used to hold a per packet state at the routing nodes, to avoid causing a broadcast storm through continuously resending the same packet through the network.

CHAPTER 5

Proof of Concept

5.1 Implementation

To show the feasibility of the approach described in Chapter 4, it was attempted to deploy the described stack on constrained devices in an IEEE 802.15.4 network.

5.1.1 Contiki

Software development for networked, embedded devices is time consuming and error-prone [90]. With rising numbers of platforms, and ever more devices of different make and model being available, it becomes increasingly difficult to maintain software, e.g. a protocol stack, for more than one platform, which in turn makes interoperability between these devices less probable. To circumvent these effects by applying the open-source approach of use, improve and share to the world of sensor nodes or the IoT in general, *Contiki* was developed and published as "operating system for tiny networked sensors" [91].

Over the time, the Contiki codebase grew from supporting *uIP* (a minimal IPv4 stack) to, at the time being, a variety of routing and communication protocols aimed at LLNs and constrained devices, most notably RPL and 6LoWPAN, as well as various RDC implementations (see Chapter 2).

Contiki also comes with a native CoAP engine, Erbium, described in Section 5.1.2 and with a hardware level simulation environment, Cooja, described in Section 6.1.1, and was thus chosen as platform for the development of the mote firmware for the implementation of the concept described in Chapter 4.

5.1.2 Erbium

Drawing from the REST implementation with HTTP-binding in Contiki [92], [93], an implementation of *Erbium*, a CoAP implementation was published in [94], and added to Contiki, as an alternative with drastically reduced power usage, being better suited for LR-WPANs.

Erbium originally implemented the infrastructure for server and client of the CoAP draft versions *03* and *07* [95], [96], with the addition of *blockwise transfers* and the *observe* option (see Sections 2.10.1 and 2.10.2). The engine was updated regularly for newly published IETF draft versions, and is at the time of writing up-to-date with the currently proposed IETF Standard for CoAP [6], while still offering *observe* and *blockwise transfers* based on the respective drafts[1] [34], [35].

Erbium underwent several updates and recently one major refactoring with removal of compatibility with all previous draft versions during the course of this work, which means that different parts of the work used different Erbium implementations and different CoAP versions.

For completeness sake it shall be mentioned that other CoAP implementations in various languages and for various platforms are already available (e.g. *Californium*, *CoAPy*, *jCoAP*, *libcoap*, *TinyCOAP* [60], [97]–[100]). Because Erbium is part of the Contiki codebase and was consistently up to date with the IETF CoAP drafts and now the proposed standard, and even implemented some non-yet-standard parts of CoAP (e.g. *blockwise transfers* or *observe* [34], [35]) it was the obvious choice for this work.

Further, the resources offered by the respective Erbium endpoints were represented as OBIX encoded XML and/or EXI messages (see Section 2.12, and extended with the IoTSyS contracts presented in [101]. Listing 5.1 depicts the response of a respective IoTSyS temperature resource.

Listing 5.1: OBIX Example Contract for a Temperature Sensor, taken from [102]

```
<obj href="iot:TemperatureSensor">
  <real name="value" href="value" val="28.21600341796875"
      unit="obix:units/celsius"/>
</obj>
```

Listing 5.2 shows an example for a message that is either received when querying a light switch for its current state, or can be sent via CoAP PUT to a light switch with a modified value to modify its state. In this example `val="false"` could be changed to `val="true"` to turn on the lights.

Listing 5.2: OBIX Example Contract for a Light Switch Actuator, taken from [102]

```
<obj href="iot:LightSwitchActuator">
  <bool name="value" href="value" val="false" writable="true"/>
</obj>
```

5.1.3 Group Communication in Erbium

Group communication following the *core-groupcomm* draft is not implemented in Erbium at the time of writing, thus some changes were added to the Erbium CoAP engine to support group communication, resulting in a prototypical implementation, where the CoAP resource could be added to one or more communication groups by listening on the corresponding IPv6 multicast addresses and default CoAP port for incoming datagrams.

[1] Unfortunately it is not clear which draft version for each is currently implemented in Erbium

Modifications were made to Erbium, so that:

1. CoAP messages could be sent as group messages to an IPv6 multicast address (in a non-confirmed way).
2. An additional callback handler type for group messages was introduced.
3. IPv6 multicast messages would be handled by the Erbium engine and delivered upwards to a previously registered service callback handler.

The changes to Erbium are listed in Listing A.1.

5.1.4 Multicasting for RPL Networks

LOWPAN_BC0

During the course of this work, the multicast routing engines presented in Section 2.8 were not available in Contiki, thus a method for multicast routing had to be found and implemented, in order to evaluate group communication for CoAP. To facilitate group communication for Contiki devices in the IoTSyS project, a multicast mechanism based on mesh broadcasting (using the LOWPAN_BC0 header defined for 6LoWPAN in [7]) was implemented for Contiki. The respective changes to Contiki are shown in Listing A.2.

LOWPAN_BC0 defines a broadcast header which is intended to be used as mesh layer broadcast facility and consists of an own dispatch code and a sequence number (to suppress duplicates and thus obviating the broadcast storm problem [103]).

For the remainder of this work, the term LOWPAN_BC0 will be used synonymously with the described flooding mechanism used as multicast alternative for CoAP group communication.

Multicast Engines in Contiki

At a later point during this work, the multicast engines *SMRF* and *MPL* (see Sections 2.8.1 and 2.8.2) became available in Contiki and were thus also used as replacement for LOWPAN_BC0. Due to the structure of Contiki, no changes had to be made to the application code to change the underlying multicast mechanism, besides disabling the hooks installed in the by LOWPAN_BC0 in the lower layers.

5.1.5 Message Construction

Message construction happens by fitting in the value to be sent into the constant parts of the message that surround the value. Listing 5.3 illustrates how the part of the message before and after the value `22.3` would thus be hardcoded, and the actual value filled in between those hardcoded parts. This was done for either XML or EXI encoding.

Listing 5.3: Example Temperature Sensor Reading

```
<obj href="iot:TemperatureSensor">
  <real val="22.3"/>
</obj>
```

This approach was deemed sufficient to prove the feasibility of the proposed concept and to show interoperability with other devices. More sophisticated approaches are expected to considerably improve the memory usage of the resulting application code, considering that the constant parts of the message have to be hardcoded in the application code, instead of constructing the XML stream on demand.

5.1.6 EXI Encoding

Due to incompatibilities with the available EXI libraries and the used schema, it was chosen to pre-encode the EXI variants of the responses before deployment and hard code them in the application. From here on, the same method as for XML messages described in Section 5.1.5 was used. Please note that only byte-aligned EXI encoding was used [9], even though this slightly diminishes the benefits of EXI, but it also reduces the complexity of message construction drastically, as concatenating bytes into a fixed structure is clearly simpler than concatenating bits.

It is worth mentioning that bit-aligned EXI stream encoding on the device *should not be generally dismissed* for this approach, as it promises vast advantages over the chosen simplification. In [66], Caputo et al. show the feasibility of a very similar approach using EXIP. The impact of EXIP on a typical Contiki device has yet to be analyzed to determine if it qualifies to be run along a CoAP stack on such a device.

5.1.7 Interaction

The scenario shown in Figure 5.1 was used to demonstrate the interaction with other CoAP-enabled devices.

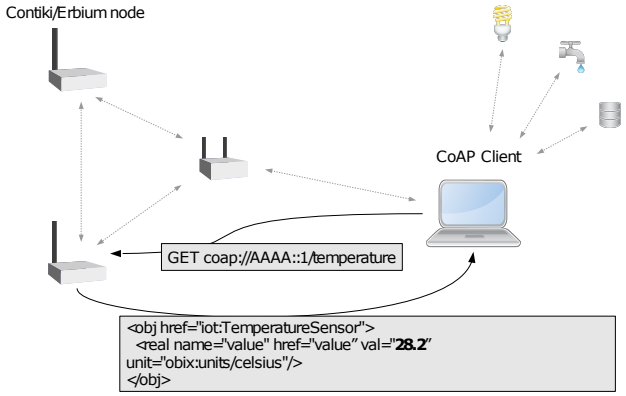


Figure 5.1: POC interaction with other CoAP devices

The constrained node offers a `temperature` resource, which reports the readout from the embedded temperature sensor as an OBIX encoded message, similar to Listing 5.3. A CoAP client would request the value and implement a thermostat logic to enable or disable actuators, depending on the current temperature value and the configured set-point. The used CoAP client

is an instance of the *IoTSyS gateway*, which enables the creation of *virtual* CoAP resources that are in fact connected to legacy HBA sensors and actuators, e.g. KNX devices [104].

The same scenario was used to demonstrate the *observe* option for CoAP `GET`, which offers the possibility to receive scarce updates about resource changes to avoid polling the value (see Section 2.10.2). With this optimization, the amount of messages being received and transmitted by the constrained device can be considerably reduced.

Additionally, group communication using was demonstrated with the scenario shown in Figure 5.2.

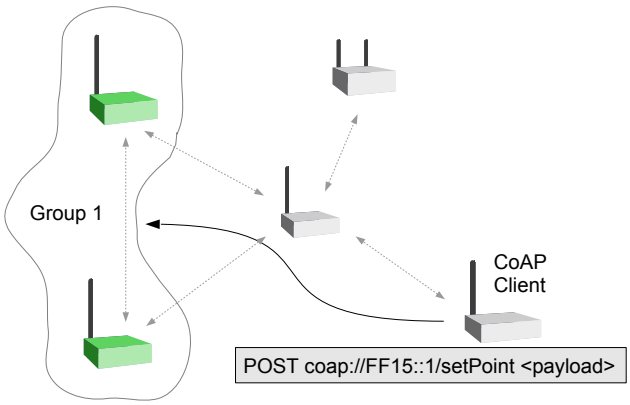


Figure 5.2: POC group communication with CoAP

Please note, that not the CoAP servers, but particular resources on these servers took part in the respective functional group.

5.1.8 Components

The proof of concept setup consists of two components:

1. A CoAP server based on Erbium offering a temperature sensor reading as OBIX object encoded as either XML or (byte aligned) EXI, depending on the submitted *accept* type, available through a CoAP `GET` request[2].
2. A border router, based on the Contiki example *rpl-border-router*[3], acting as RPL root and as 6LoWPAN border router, allowing conventional IPv6 devices to reach into the 6LoWPAN network to interact with the CoAP endpoints.

The components were run on separate nodes: Two CoAP Zolertia Z1 motes were used to deploy a server on each of them and a Zolertia Gateway[4] was used as border router. For details

[2]https://github.com/bipson/contiki/tree/2a2a9fca8/examples/er-rest-example-sens

[3]https://github.com/contiki-os/contiki/tree/79e6514c80/examples/ipv6/rpl-border-router

[4]The Zolertia Gateway is an embedded PC with an in-built Z1 mote, that is well suited for usage as router/gateway to a IEEE 802.15.4 WSAN

on the used devices please refer to [105]. The version of Contiki used[5] supports the CoAP draft version 13 [106].

5.2 Evaluation

5.2.1 Contiki

Contiki as a project is due to its academic origins subject to frequent changes, as well as occasional experiments and naturally also the subsequent bugs. Another effect is that some implementations might not be 100% up-to-date with the underlying standards from the first day on, while other features and parameters are only tested in small scale developments, thus not yet completely refined and might easily encounter corner cases. The structure of the project is mainly due to its lean architecture for constrained devices, but also means that development of certain (experimental) features can be cumbersome.

The efforts to bring Contiki into a more stable and productive state are tangible and expected to progress quickly, and this is necessary if Contiki is intended to be used for larger scale installations and experiments, also for those with academic background, or even real-world applications.

5.2.2 Memory Constraints

The memory constraints of the chosen platform are very rigid, where the main issue in this respect is the content of potential messages residing in the memory of the constrained device due to the chosen implementation. This would easily attribute to 100 Bytes in ROM for one simple, non-nested data-point. It is apparent, that for XML messages the majority of the message content is redundant and, especially on constrained devices, a waste of precious memory and power, as sending larger messages presumably uses more power than sending smaller ones. Further, in the tradition of information theory, every byte of the used XML message conveys only little information, which hurts even more if the memory is so precious on constrained devices.

Serializing the XML message upon request would in principle improve this, but implementing full-fledged XML stream serialization on a constrained device seemed to be infeasible. Although using EXI encoded messages already relaxes this problem considerably, a more convenient alternative to the used construction of messages is still desirable and could reduce pressure from memory constraints even further [63], [68].

Figure 5.3 depicts the impact of the changes to Contiki and Erbium to make multicasting and group communication possible (see Listings A.1 and A.2).

The changes in the network stack due to LOWPAN_BC0 and the changes to the Erbium engine only attribute to an increased binary size of under 2%. The footprint of a group communication handler in Erbium in the proposed structure only causes an increase of well under 0.5%. On the other side, multicasting results in a $\tilde{1}1.5\%$ smaller binary for the client compared to unicasting with four individual messages, also saving the state machine of the conventional CoAP request, as unicasting uses “Acknowledged” requests.

[5] https://github.com/contiki-os/contiki/tree/79e6514c80

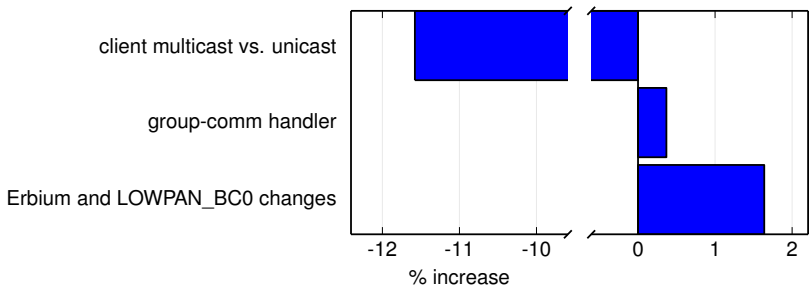
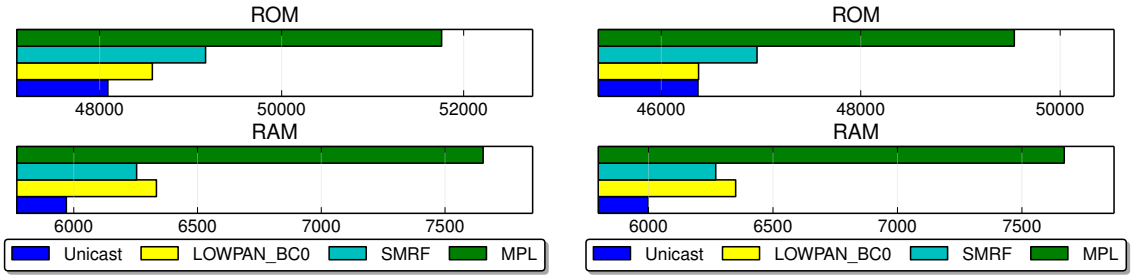


Figure 5.3: Binary footprint for multicast vs. unicast

The comparisons for RAM and ROM usage of the different multicasting engines are shown in Figure 5.4. The shown results are from compilations without ContikiMAC, as MPL together with ContikiMAC and Erbium would not fit on a Z1 mote.

It shall be noted, that for the “server” node only the used endpoints were compiled, i.e. the group communication/multicast endpoint is missing from the unicast binary, and the unicast endpoint missing from the binary used with multicast engines due to memory constraints.



(a) memory footprint client

(b) memory footprint server

Figure 5.4: Memory footprint comparison for multicast engines

All simulated engines are comparable regarding their binary sizes, a part from MPL, which once more shows a worse performance compared to the other engines/mechanisms, especially its RAM usage is concerning. The shown figures also make it obvious, why using ContikiMAC is not possible in conjunction with MPL and Erbium, as they together would easily exceed the 8 KB RAM available on the Z1 platform for the “client” and “server” nodes.

For all other mechanisms, unicast consistently has the lowest usage of RAM and ROM. This is only partially due to the Erbium group communication additions, following the results from Figure 5.4, and apparently mostly owing to the multicast routing engines/mechanisms.

5.2.3 Routing

Reaching 6LoWPAN devices via their IPv6 address from the IPv6 device that is directly connected to the border router device, e.g. via Universal Serial Bus (USB) and thus running the

tunnel interface into the WSAN is straightforward and works reliably out of the box, due to the necessary routes being automatically added by *tunslip6*. For other IPv6 capable devices, even if residing in the same link-local networks the device hosting the tunnel into the WSAN, the same approach would make it necessary to add the respective route(s) via this device as gateway *manually* on all other devices requiring access into the WSAN.

Stateless autoconfiguration in IPv6 networks would typically happen via a router advertisements (e.g. by *radvd* [107]) and/or using Neighbor Discovery Protocol (NDP) to advertise the local prefixes and default routes to devices on the same link [108]. In this setup, whenever a device is trying to reach an IPv6 address outside the local network, it will address this request to the default gateway previously advertised. At the time of writing though, advertising *additional* routes via router advertisements proves to be cumbersome[6], making stateless autoconfiguration only work for this scenario if the gateway into the WSAN is at the same time *also* the default gateway out of the local network, which is not always desirable.

Additionally, 6LoWPAN requires a prefix length of 64 bit for the addresses of devices inside the WSAN [7], so does stateless address autoconfiguration for IPv6 [108], which will most certainly result in 6LoWPAN networks not being nested inside networks with stateless address autoconfiguration (i.e. the same prefix length).

Thus, a WSAN *inside* an existing IPv6 infrastructure featuring stateless autoconfiguration is not reasonably possible, and such a WSAN should be setup at least one level higher, requiring a 64 bit prefix of its own to make the devices inside the WSAN globally available.

5.2.4 Debugging

Development of application firmware on Contiki motes quickly becomes very time consuming, considering the waiting time for a device to become available through its respective gateway and the scenario requires reachability of the respective device from a device outside of the WSAN. The author thus strongly recommends Cooja for the development process of Contiki applications, as it can considerably speed up testing of applications or protocols, prior to moving to deployment on real devices.

5.2.5 *Observe* Content-Type

At the time of execution of the proof of concept, Erbium offered a decent implementation of the *observe* functionality (cf. Section 2.10.2), with the necessary considerations for the constrained devices it is intended for. Unfortunately, when observers would register with a CoAP server via the `OBSERVE` method, the server would not inherently track the *content-type* sent by the observer via the *accept* option. Instead, every registered observer would receive the same message in the same encoding whenever a notification is triggered. As the list of observers and the notification of those are integral parts of Erbium (see Listing 5.4), it is considered appropriate to implement the respective functionality rather at this level, instead of botching it on top of it in the application itself, even more so, because it could supposedly be useful for other applications as well. It is

[6] On presumably most systems running a more or less recent Linux kernel ($\leq$ 3.14.4) the according option is disabled by default (`net.ipv6.conf.all.accept_ra_rt_info_max_plen` is set to `0`).

not intended to diminish the complexity of such a solution, and a potential solution should still be able to avoid constructing excrescent responses.

Listing 5.4: CoAP call for observer notification

```
void coap_notify_observers(resource_t *resource, int32_t obs_counter, void
    *notification);
```

That being said, another possibility to overcome the lack of this inbuilt functionality, would be to restrict the application to a single encoding, and enforce this restriction on the environment/interacting devices. Considering the fact that highly constrained devices are used, it is safe to assume that the use case for such a device is very specific, thus it might be little to no issue if only a single encoding is available for the *observe* functionality. In consideration of the issues described in Section 5.2.2, this could even extend to all the other verbs offered by such a device besides *observe*.

Please note that with the rewrite of Erbium at a later point, Erbium in fact gained the capability to track the content type per observer[7], which was not evaluated.

[7] https://github.com/contiki-os/contiki/tree/14aedab/examples/er-rest-example

CHAPTER 6

Simulations

6.1 Introduction

In order to analyze the stack under more complex scenarios and without having to rely on cumbersome deployments on real devices, simulations of the respective networks are executed in **Cooja** (see Section 6.1.1). The goals of these simulations are primarily to measure the **latency**, **power consumption impact**, as well as **reliability** of CoAP under various scenarios and compared to other bindings. Particularly energy consumption is of utmost importance for a prospective application protocol for constrained devices. Another important question that will be covered by the simulations are the **limitations** of CoAP, to show if CoAP is viable for real world applications.

Additionally, detailed measurements of the small effects occurring in larger networks are impractical and in some cases even close to impossible to be reliably executed on real devices in real time. Another advantage of simulations, is that the generated output and radio traffic of every device in the network is available for immediate and/or if recorded for later inspection, which in return avoids cumbersome and error-prone collection of their data, as the data are spread over multiple devices throughout the network. In the worst case, trying to retrieve the distributed data might affect the measurement results.

The results are measured comparatively, which means they are gathered and compared against the competitive binding or scenario, offering insights about benefits and drawbacks of the observed scenario without being impacted by side effects occurring due to the measurement framework. This requires that the way the measurements are taken are comparable or even identical.

As it will be shown, CoAP offers vast advantages over comparable bindings, while multicasting offers considerable improvements regarding energy consumption and application code complexity for CoAP group communication. The limitations of the used stack regarding network sizes look promising.

6.1.1 Methodology

Cooja

Contiki is distributed together with Cooja, a network simulator that, in difference to other typical network simulators, can and usually does simulate the participating nodes on the hardware level [10]. Its main purpose is the simplification of debugging of complex interactions of single devices inside a network of such devices, without the need to attach to all the participating devices to collect their real-time debugging output via hardware interfaces. In general, exactly the same code will be used for simulated nodes that is else used for the devices in a real deployment. A created node *type*, defined through its specific firmware and hardware make and model, will be placed in relative distance to other nodes on a virtual plane (see Fig. 6.1), and have various of its interfaces made available through the respective Cooja plugins.

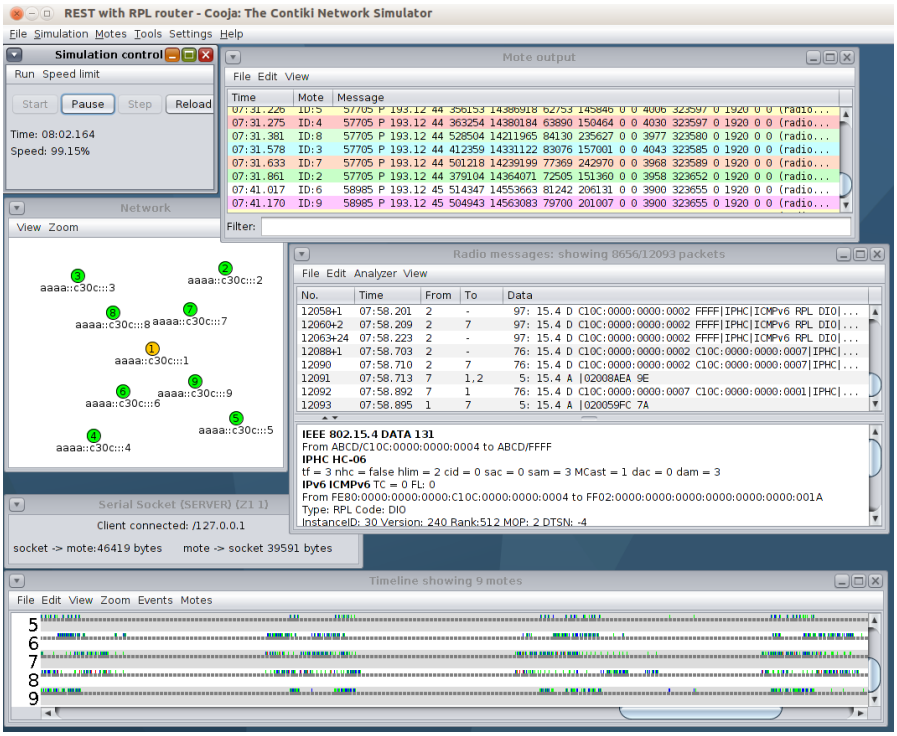


Figure 6.1: Screenshot of Cooja with running simulation

One of the main drawbacks of Cooja compared to other designated network simulation tools is the **lack of native support for collection of network related metrics**, e.g. network saturation, collision rate, or dropped packets. This information must thus be abstracted and measured as higher level metrics, for example dropped messages at the application layer instead of low level packet drops/collisions.

Additionally, Cooja offers a **simulation scripting facility**, that enables simulations and interactions with devices to be automated, in turn making automatic and thus repeated simulations possible when the corresponding simulation is run in *headless* mode. This mode is available

when Cooja is started via the `-nogui=<csc-file>` argument. This also reduces the total run-time of simulations by a considerable amount.

While the collection of the occurring console output of all motes is natively supported by the Cooja test scripting facility, the collection of radio traffic is not. Unfortunately, radio packet dumps proved to be a valuable source for latency measurements with certain simulations, thus a *Headless Radio-Logger* plugin was adapted from an identical named Cooja plugin, which was written for the same purpose but for an older code base by Cetic [109].

Powertrace

To measure the energy consumption of an application running on an embedded device is not trivial, in particular if the resulting firmware is expected to use rigorous power saving methods, like disabling not used parts of the device or switching to lower CPU power states whenever possible. Yet energy consumption is maybe the most important metric for the evaluation and comparison of embedded firmware or components.

This is also true for applications built using Contiki, thus powertrace was presented in [110]. Powertrace uses internal counters which are updated through *timecapsules*, i.e. instructions surrounding energy consuming blocks of code for every device, e.g. a block of instructions which enable the radio transmitter to send data, and thus track the duration of this block, in corresponding numbers of elapsed CPU cycles. These counters are summed up and printed to the mote console output in a configurable interval. The developer can use powertrace by just adding a few lines of code to the mote application, i.e. by adding the powertrace *app* to the application (add `APPS += powertrace` to the Makefile), including the respective header (`powertrace.h`) and starting the respective process with `powertrace_start(<interval in ticks>);`.

The mote output that is triggered with the given interval showing the current counter values, has the structure shown in Listing 6.1.

Listing 6.1: Powertrace Output and Structure

```
5225 P 193.12 3 28856 1281642 9898 10120 0 0 5749 319481 720 2740 0 0 (radio
    1.52% / 1.06% tx 0.75% / 0.22% listen 0.77% / 0.84%

<time> P <node rime-addr> <sequence-#> <SUM_cpu> <SUM_lpm> <SUM_transmit>
    <SUM_listen> <SUM_idle_transmit> <SUM_idle_listen> <cpu> <lpm>
    <transmit> <listen> <idle_transmit> <idle_listen> (<radio statistics
    ... >))
```

Although the obtained numbers can be used to estimate the energy consumption of a device with the knowledge about the consumption in the respective states [110], the obtained counter values without a conversion into energy consumption should be regarded as dimensionless and are not to be taken as estimate of the expected life-time of a device. Nevertheless, they are a good way to relatively compare the impact on power consumption of different operation modes.

Another related method is the *powertracker* plugin for Cooja, which can measure the radio on/transmit/receive/interfered times of a network. The plugin was not considered in this work.

6.1.2 Measured Metrics

Latency/Response Time

Depending on the simulation two different methods are used to measure the latency of requests:

1. By tracing the timestamps of the respective packets in the **radio log** that indicate beginning and end of a request through patterns.
2. By tracing the timestamps of the **mote output log** (originating from the simulated mote applications) of sending and receiving motes, which indicate beginning and end of a request.

Although measuring the latency through tracing the radio log is suspected to be more accurate than tracing the mote output log, it is also considerably more complicated and error prone, thus which method is used depends on the simulation setup and if the respective output is available.

While tracing the mote output log is trivial, tracing the radio log involves comparison of the timestamps of packets as they pass a certain node in the simulated network[1], i.e. **the difference of the timestamps of (1.) the packet indicating the beginning of a request and (2.) the packet indicating the end of a request**. Depending on the particular setup, and if the simulated transmission is of a request-response type or consists only of a one-way request, this would either be the time difference between

1. the (first occurrence[2] of the first fragment of the) message being sent/forwarded by the observed node to the next node in the message's path and the (last occurrence of the last fragment of the) response being again received at the observed node or
2. the (first occurrence of the first fragment of the) message being sent/forwarded *by* the observed node and the (last occurrence of the last fragment of the) message itself being received at the intended node/recipient.

For simulations that involve multiple, quasi simultaneous requests, i.e. a request to multiple recipients, the measurements are taken in a similar fashion: the time difference between the (first fragment of the) first message indicating the beginning of the series and the (last fragment of the) last message indicating the end of the series is measured. The particular method used will be explained for every setup in detail.

Please note, that for topologies susceptible to latencies above the generally used interval of **10 seconds**, the simulations in question are rerun with a higher interval to gather meaningful latency, because reliably detecting beginning and end of requests in the above mentioned manner would be considerably more complex and overlapping requests would most certainly influence each other.

[1] Typically this would be the node generating the request, but for some simulations the requesting node is *outside* of the simulated network, thus the packets passing a representative node inside the network are observed.

[2] The *first* occurrence is taken as reference, because with certain operational modes of a WSAN (e.g. when RDC is applied) messages will be **resent** multiple times. For the same reason, the last occurrence of the last fragment of the message arriving at the destination is used.

The latency numbers will typically be displayed as canonical **box-and-whisker diagram**, that illustrates

- the average,
- the median (50th percentile, Interquartile Range (IQR) or 2nd quartile) as box,
- the whiskers (calculated as lower end of IQR - 1.5 * IQR and respectively upper end of IQR + 1.5 * IQR) that demarcate the 1st and 3rd quartile, as well as
- outliers that do not fit inside the three quartiles.

Drop Count

Mainly due to shortcomings of the used simulation framework regarding network metrics, but also considering the scope of this work being concerned with the performance of the application protocol, packet drop and collisions at the physical layer are not considered for measurements using the simulation environment, which are caused at and mostly also resolved by the lower levels of the protocol stack. Thus the "dropped messages" at the application layer are counted instead. Depending on the setup, a dropped message would be registered as such whenever:

1. The requesting node would not receive the expected payload within a given time frame after the request is sent.
2. The resource node would not receive the request within a certain time frame after the request is sent.

For all setups a timeout of **10 seconds** is assumed.

Power Consumption

The power consumption is measured by using the powertrace infrastructure of Contiki. With this infrastructure not the power consumption itself is measured, instead the cycles spent in certain (energy consuming) states are counted, e.g. *for how many cycles the radio was enabled, listening actively for incoming radio transmissions* (see Section 6.1.1).

Measurement The numbers for power consumption are collected and periodically written to the console by the device itself. The output already contains the sums for various values, like active CPU time, or radio transimission time. In Figure 6.2, the numbers collected for a device responding to periodic requests are plotted against the timescale.

As every simulation of a RPL network is subject to a *startup delay*, until all devices are reachable under their designated IPv6 address and the vast number of network configuration messages ease down (causing increased network traffic unrelated to the simulation objective), only the last n readings of a powertrace log are considered for the simulation results (the first readings are dismissed), and the difference between the last and first reading scaled to a per-minute value (see Eq. (6.1)).

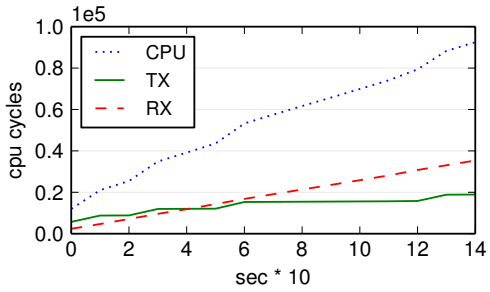


Figure 6.2: Visualization of data collected by powertrace

$$cpm = \frac{cpuSum_x - cpuSum_{x-(n-1)}}{n * \frac{1}{m}} \tag{6.1}$$

where

cpm	...	cycles per minute
$cpuSum_x$	...	sum of CPU active cycles at a certain measure point x, yielded by powertrace
n	...	number of readings considered
m	...	number of readings logged per minute by powertrace

Three of the available values will be observed:

CPU time The time spent while CPU was *not* in power saving mode (i.e. active)
Radio TX time The time spent while radio was actively switched on for sending
Radio RX time The time spent while radio was actively listening for incoming transmissions

6.1.3 Topologies

In Figure 6.3, an example topology is shown.

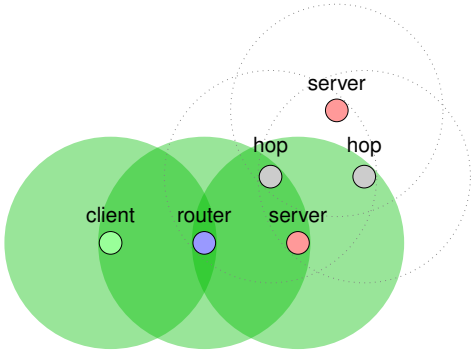


Figure 6.3: Example topology for simulations

The circle around a node indicates its individual range, which determines if another node is visible to this node or not. The ranges of some nodes are highlighted in green, to make the

resulting path easier to spot and have no special meaning for the simulation apart from that. Intermediate "hops" between nodes are forced by *placement* (i.e. nodes can only communicate over the nodes inside of their transmission radius).

For all simulations, the Unit Disk Graph Medium – Distance Loss (UDGM) is used, with the parameters shown in Table 6.1.

Transmitting Range	50
Interference Range	50
Success Ratio TX	100%
Success Ratio RX	100%

Table 6.1: Cooja settings for the simulations

Four types of nodes are used throughout the simulations (with slight variations):

- **Server** The node offering the queried resource/endpoint. Multiple servers might be deployed in one topology, e.g. when multiple resources are queried simultaneously. The used application code is based on either the Erbium example server, or the REST example for HTTP, both part of the Contiki examples[3]. Please note that the examples in Contiki changed during the course of this work, thus the used code is partially based on different versions of the examples.
- **Client** This node issues the periodic requests, and receives the response, respectively. Various clients are used throughout the simulations, usually based on the Erbium example client of Contiki. For some simulations, external clients are used, based on the *Californium* example client[4], or using *cURL* for HTTP [111].
- **Router**[5] This node acts as border-router between the simulated WSAN and the "outside world". More importantly, it also acts as the RPL DODAG root and initiates the construction of the RPL DODAG. As application, the IPv6 *rpl-border-router* from Contiki is used[6].
- **Hop** This node has no special purpose, besides acting as a intermediate "hop" between adjacent nodes.

6.1.4 Notes on Radio Duty Cycling (RDC)

The operation of a WSAN is only justifiable with RDC applied to conserve considerable amounts of power, vastly extending the battery lifetime, and thus the maintenance intervals, of the participating nodes [12]. Hence, for all simulations, RDC is enabled and *ContikiMAC* is used as RDC enabled link-layer protocol, if not explicitly stated otherwise. Please note, that the *rpl-border-router* application always disables RDC for the node it is being run on. This influences the

[3] *rest-example* and *er-rest-example* in https://github.com/contiki-os/contiki/tree/master/examples

[4] https://github.com/mkovatsc/Californium/tree/767ba21/cf-client/src/main/java/ch/ethz/inf/vs/californium/examples

[5] For some simulations the "router" is replaced by a RPL "root" that acts as RPL DODAG root without the gateway functionality, e.g. when the border-router would clash with the used multicast engine.

[6] https://github.com/contiki-os/contiki/tree/a9e7bea/examples/ipv6/rpl-border-router

message paths of the presented topologies by a considerable amount if the messages are passing the border-router, and thus will be singled out where it presumably affects the results.

6.2 HTTP vs. CoAP

The straightforward approach of bringing RESTful interfaces to WSANs involves deploying an IP/TCP/HTTP stack and the respective resources on the devices forming the WSAN. Due to its design for constrained devices, CoAP is expected to perform superior compared to HTTP, at least in terms of energy consumption, which is why a comparison of CoAP and HTTP considering their applicability for WSANs is carried out. Besides the energy consumption, how CoAP performs in terms of packet delivery ratio compared to HTTP is unknown, and the concrete values for all compared metrics are of interest.

It is attempted to deploy two "identical" resources on both bindings, that offer the exact same payload through the same HTTP/CoAP verb, and thus model a comparable, periodic request/response scenario for both with the same surrounding conditions.

Server The used servers are based on the HTTP or Erbium example servers of Contiki. Both were adapted to offer the same static payload behind the same resource URL under the respective binding[7].

Client Both clients, HTTP and CoAP, were not simulated, but executed as clients external to the simulation, using either the *Californium* example client[8] or *cURL* [111]. An attempt to deploy an HTTP client on a Contiki node failed, hence both respective clients are run outside of the network. A script was used to issue either continuous requests for the energy consumption measurements or exactly 10 requests for the latency measurements.

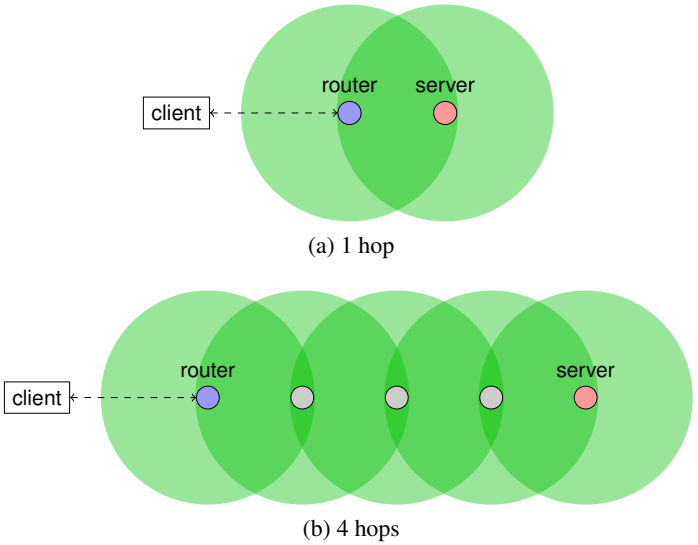
6.2.1 Power Usage

To compare the impact of CoAP to the impact of HTTP on the power usage of a constrained device, the topologies shown in Figure 6.4 are repeated with variable payload sizes of **2**, **64** and **128 bytes**. For a description of the node types, see Section 6.1.3.

The power consumption is measured at the respective *server* node. Figure 6.4a shows the simpler "single hop" topology, where the server node is placed next to the router node, so that the router forms the only hop between the client and the server. Additionally, the topology shown in Figure 6.4b with three additional hops placed between server and client is simulated, to compare how both binding's power usage is impacted by this change. This topology will be further referred to as the "4 hops" topology.

[7] https://github.com/bipson/contiki-simfork/blob/33ff93867d/examples/er-smart-meter/er-smart-meter.c *and* https://github.com/bipson/contiki-simfork/blob/33ff93867d/examples/rest-example/rest-server-example.c

[8] https://github.com/mkovatsc/Californium/tree/767ba21/cf.-client/src/main/java/ch/ethz/inf/vs/californium/examples



(a) 1 hop

(b) 4 hops

Figure 6.4: Simulation topologies CoAP vs. HTTP - power consumption

Results

Figure 6.5 at first glance bares the *vast* power savings of CoAP over HTTP for all measured topologies and variables. Please note that the numbers that were gathered from simulations *not* using RDC offered little additional insight and are thus not shown.

For most measured metrics, even the best numbers for HTTP are worse than the worst numbers yielded by CoAP. The numbers for radio transmit and radio listening time are of particular interest, as both are considered to contribute significantly to the energy usage and thus potential savings of a constrained device.

In all figures, a fall-off is distinguishable for HTTP and payloads of 128 bytes, where the numbers would be expected to be much higher following the previous trend, as if the data were cut off. This is due to the fact, that with HTTP for this payload size a significant amount of requests were *aborted* when the response took longer than the request interval of 10 seconds and thus timed out. This becomes apparent if the results for dropped messages in Section 6.2.2 are taken into consideration.

On the other side, the numbers for CoAP show an unexpectedly sharp increase from 64 bytes to 128 bytes. This can be explained with **CoAP *blockwise transfer*** (see Section 2.10.1) taking place, resulting in twice as many messages between client and server than for smaller payloads[9].

The numbers for *CPU active time* in Figure 6.5a show a distinguishable increase in CPU usage when the hops change from 1 to 4 for all payloads, but more prominently so for HTTP. The indicated trend would suggest that the CPU usage does increase *less* for CoAP per hop as it does for HTTP. It is unclear why an increase in intermediate hops results in raised CPU usage,

[9] Although *blockwise transfer* is not part of the CoAP specification [6], it is implemented in Erbium nonetheless and the default payload chunk size set to **64 Bytes**. In the presented case, this would result in two subsequent 64 Byte large messages instead of a single 128 Bytes message. This is not to be confused with IEEE 802.15.4 fragmentation.

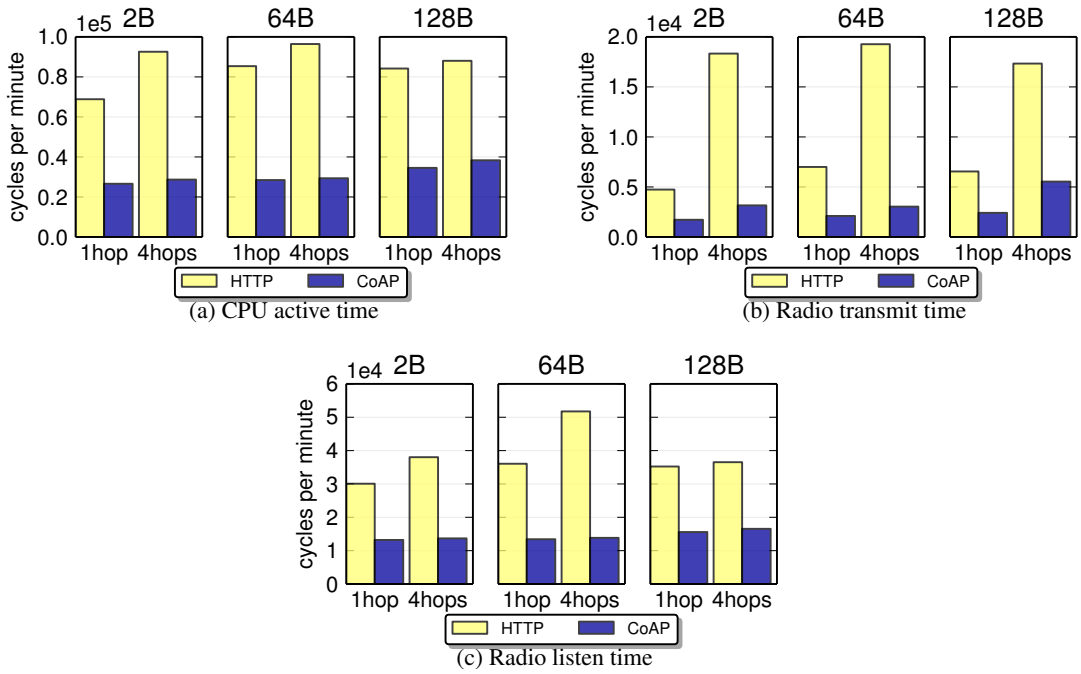


Figure 6.5: Power consumption HTTP vs. CoAP - RDC enabled

and effects from the neighboring hop in the 1 hop scenario (i.e. the border-router) being RDC-disabled is suspected to influence the amount of CPU usage. Increasing the payload size also has a visible impact on the CPU usage, although more consistently for CoAP, where HTTP on the other side only shows a distinguishable increase from 2 bytes to 64 bytes payload.

Figure 6.5b shows the increase of *radio transmission time* with increasing hop counts, with a remarkably higher factor for HTTP than for CoAP. As the same payload is sent for all hop counts, it is assumed that this is again caused by the adjacent node to the server being the RDC-disabled border-router, which will largely suppress retransmissions by the server, thus lead to decreased radio transmit times. As HTTP generally requires considerably more individual packets being transmitted and received to respond to one request, this affects HTTP more than CoAP. Larger payloads do not generally correlate with active transmit time, although two exceptions are visible: HTTP shows a distinguishable increase for lower payloads and the single hop topology, for which the reason is unclear, and CoAP shows a clear increase for 128 bytes payload and the 4 hops topology. It is assumed that this is due to *blockwise transfer* taking place, where fragmentation happens end-to-end. With end-to-end fragmentation, the client will trigger a new request for the next block only after the previous block was received and a remaining payload indicated, which is presumably further boosted by the effects of the RDC-disabled border-router.

The time the radio was *enabled for listening* is only slightly impacted by different payloads and topologies for CoAP, in contrast to HTTP, as shown in Figure 6.5c. For HTTP, the required time listening on the radio seems to correlate very closely with both, increasing hops and increasing payload sizes, while dropped messages presumably disrupt the trend for 128 bytes. The

increase is assumed to result from the higher number of individual transmissions being sensed by the device and resulting in the device radio being kept awake to receive the packet [12]. For CoAP, only a slight increase, presumably from *blockwise transfer*, at 128 bytes is noticeable.

As already stated, the presented results are from simulations **with RDC enabled**. To compare the impact of RDC for both bindings, Figure 6.6 shows the difference in cycles per minute when enabling RDC for a payload of 64 bytes.

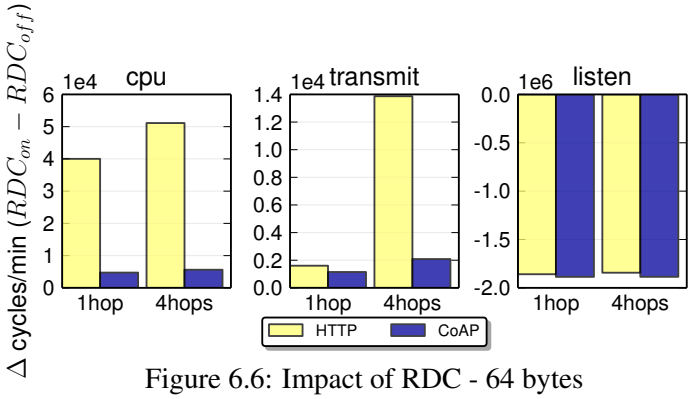


Figure 6.6: Impact of RDC - 64 bytes

As expected, enabling RDC will *increase* the power spent for CPU and radio transmissions slightly, for *vast savings* regarding total time spent listening on the radio. This is valid for both bindings and all topologies (note the different scales in the figure). Nevertheless, using CoAP results in slightly better savings for radio listening time, and *considerably* less negative impact regarding CPU usage and radio transmit time when using RDC.

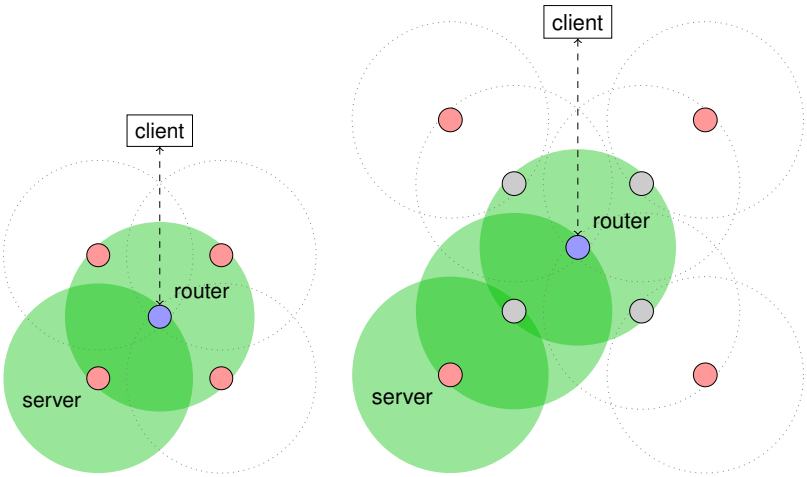
Increasing the hops from 1 to 4 does result in additional power usage for CPU and radio transmissions for all bindings, mainly because the adjacent hop for the server in the single hop topology does not use RDC. The power savings for time spent listening on the radio are not significantly impacted by the changed topology.

These results suggest that CoAP copes with RDC considerably better than HTTP, which is coherent with the conceptual foundations of CoAP, and at the same time confirms its suitability for constrained environments in general. On a broader scale the results also hint at a better support of CoAP for the techniques used in constrained environments to reduce power usage, due to how CoAP message transactions are designed and its modest use of resources, in particular its low radio transmissions' overhead.

6.2.2 Latency

To investigate the latency related behavior of CoAP to HTTP, and to exploit certain behavior of the underlying network, the setups of the following simulations are slightly modified from the setups in Section 6.2.1.

The Figures 6.7a and 6.7b depict the simulated topologies, where either **one**, **two** or **four endpoints** were queried. All nodes not participating in the simulated scenario were *removed*. For the sake of brevity the respective figures without the removed nodes are not shown, but the topologies can easily be derived from the shown figures.



(a) 1 hop (b) 2 hops

Figure 6.7: Simulation topologies CoAP vs. HTTP - latency

One or multiple *servers* were simulated offering the same static payload of either **64** or **128 bytes** and every simulation repeated 5 times. If multiple servers were queried, the requests were issued at the same time[10].

The latency for a request was measured at the first available link in the simulated network, i.e. between the border-router and the adjacent hop (see Section 6.1.2).

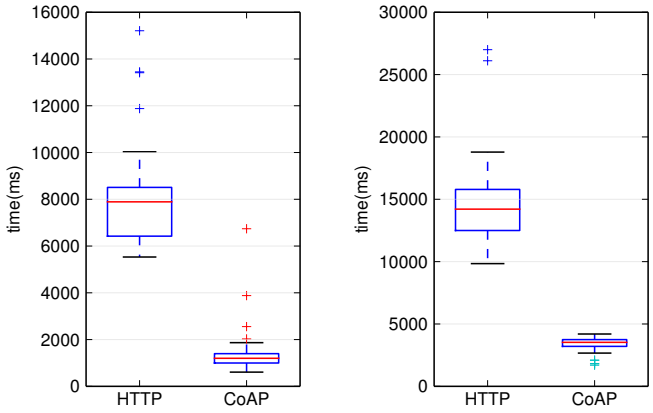
Results

The comparison between HTTP and CoAP shows a distinctive difference in observed latency for both bindings, as can be clearly seen in Figure 6.8, which displays the distributions of the collected numbers for the **2 hops setup** with **2 and 4 parallel requests** for **128 bytes payload**. Please note the different scales for the y-axis in Figures 6.8a and 6.8b. The numbers for the additional topologies and parameters are not displayed as box-and-whiskers diagram, because in the authors opinion they offer little additional insight.

The average latencies of both bindings are distinctly different, close to a factor of 8 for the two request topology, but decreases to approximately 3.5 to 4 for four parallel requests. We will look more closely at the averages over all topologies and setups further down. By comparing the quartiles of both bindings, they appear to be very similarly distributed in relation to their respective average latency. In fact, the second quartile of CoAP is typically slightly broader in respect to its median, but tends to have less outliers.

Figure 6.9 displays the average of the measured latencies for all executed simulations, illustrating the increase in latencies for additional parallel requests.

[10]Technically it is not possible to send two or more requests at *exactly* the same time, and a *best effort* strategy was used instead, i.e. requests would not wait for a precedent request to be sent.



(a) 2 requests

(b) 4 requests

Figure 6.8: Distribution for latency for HTTP and CoAP, 2 hops, 128 bytes

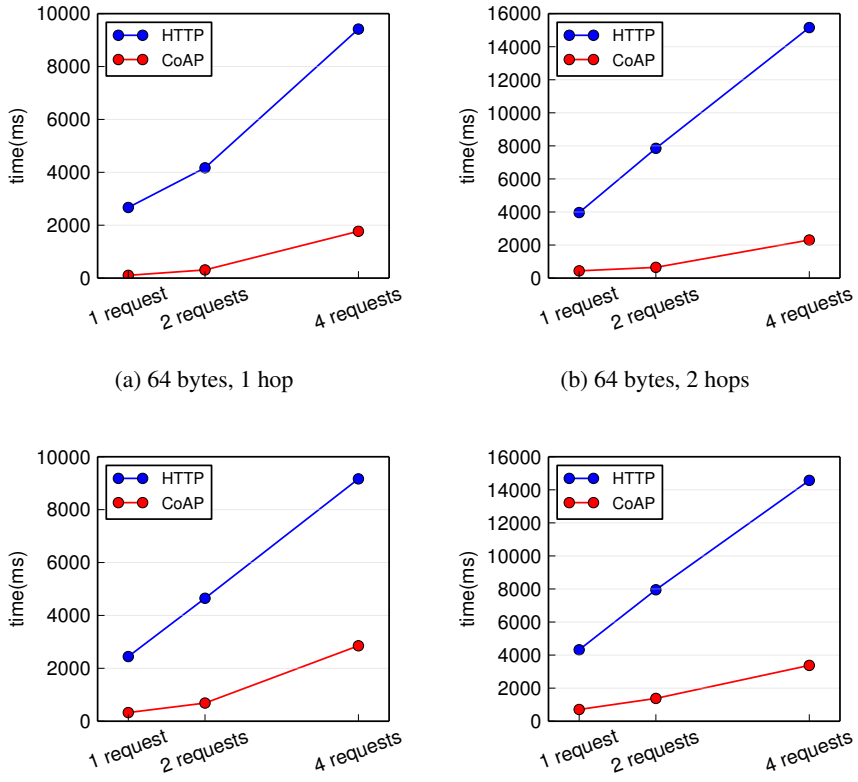
For the numbers displayed in Figure 6.9a, both bindings show a similar, approximately linear growth with increasing numbers of requests. While the growth between a single and two parallel requests seems to be moderate, it slightly increases between two and four parallel requests for HTTP but not for CoAP. For HTTP, the growth between a single and two parallel requests is slightly lower than between two and four parallel requests, which is also discernible in the figures as being above linear growth. For CoAP, the growth from the single request setup to two parallel requests is as also slightly lower than it is for the interval between two and four parallel requests. In comparison to HTTP, where the difference of growth between the two intervals is of factor 1.75, it is of factor 3.57 for CoAP, which would indicate that CoAP is impacted more by the additional requests[11].

Figure 6.9b shows a slightly different outcome: The growth from the setup with two parallel requests to four parallel requests seems to be *below* linear growth, which is assumed to be a result of the variation of the results. Although an increased number of repeated simulation runs would counteract this effect, they were abandoned because of the unreasonable execution time of every single run. For CoAP on the other hand, similar to the single hop setup, the growth is above linear growth, increases between both intervals by a factor of 3.94, approximately the same factor as for the single hop setup.

When looking at the numbers in Figure 6.9c, little has changed: the HTTP binding will be at close to linear growth with additional parallel requests being issued, while the growth for CoAP is subject to an increase by a factor of 3.

Figure 6.9d seems to yield very similar results as seen in Figure 6.9b, with almost identical

[11] Please note that the growth factor typically will not be proportional to the number of requests added, as a request always has a latency on its own, and additionally (parallel) requests will not simply add up, but *delay* other ongoing requests by a certain factor. For the same reasons, the factor of both bindings for the same "step" are not comparable, as the latency of the network itself/of the single request will impact the numbers more for the binding with the lower average.



(a) 64 bytes, 1 hop

(b) 64 bytes, 2 hops

(c) 128 bytes, 1 hop

(d) 128 bytes, 2 hops

Figure 6.9: Average latencies for HTTP and CoAP

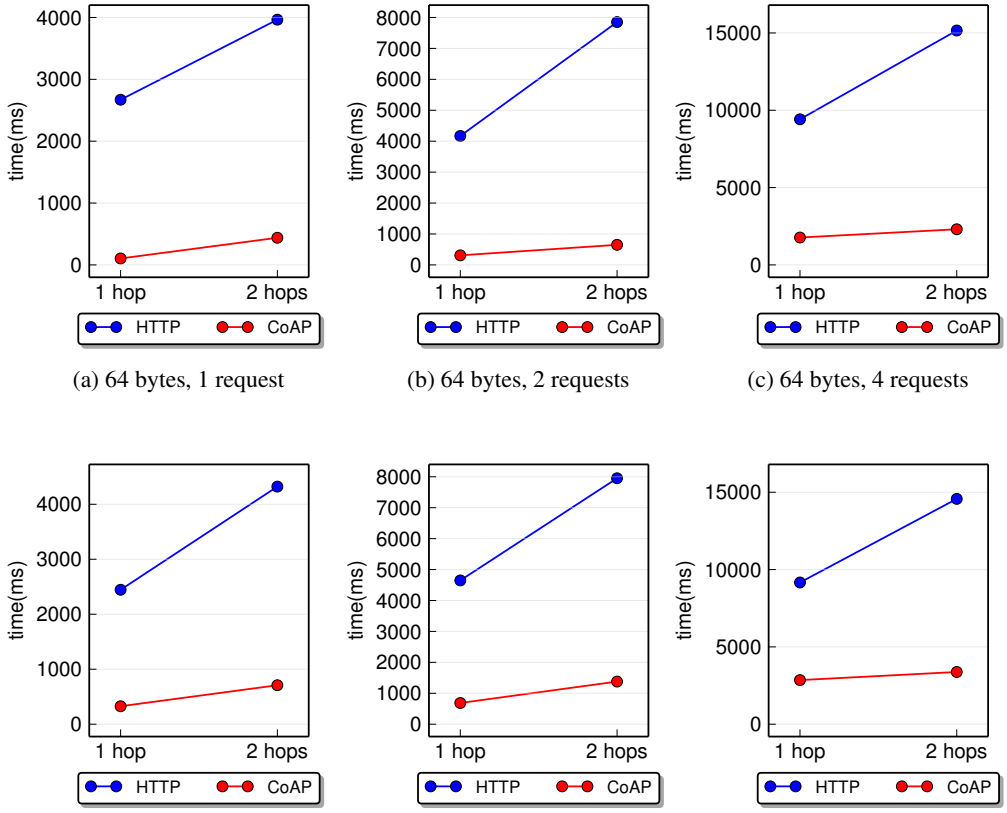
outcome for HTTP, but with a different offset for CoAP, and also with a drastically decreased factor (1.49) of change in growth.

Although the latency for CoAP is throughout significantly better than for HTTP, the factor of change in growth between two and four requests for CoAP is slightly concerning, while it is assumed that CoAP grows significantly more because it starts from far lower numbers and thus the additional requests have a stronger impact.

For additional insight, Figure 6.10 shows a different grouping for the same data, depicting the trend for increasing hops. Please note that the figures in Fig. 6.10 have a different scaling.

Looking at the simulations with 64 bytes payload (Figs. 6.10a to 6.10c), again an overall advantage of CoAP is apparent for the absolute latency, as well as for the trend for additional hops[12]. The steepness of the trend seems to change slightly for the different numbers of parallel requests (cf. Fig. 6.10c and Figs. 6.10a and 6.10b for HTTP), which seems to be still inside the fluctuation range.

[12]In this particular case having more than a single hop in the path also means having a hop in the path that uses RDC, while the single hop in the single hop topology is in fact the border-router, which does *not* use RDC.



(a) 64 bytes, 1 request (b) 64 bytes, 2 requests (c) 64 bytes, 4 requests

(d) 128 bytes, 1 request (e) 128 bytes, 2 requests (f) 128 bytes, 4 requests

Figure 6.10: Average latencies for HTTP and CoAP, by hops

Figures 6.10d to 6.10f for a payload of 128 bytes show a very similar picture, although the trend seems to be steeper for CoAP, and the absolute numbers distinctively higher compared to 64 bytes payload, where for HTTP an effect by the increased payload is not discernible. The numbers for CoAP seem to be negatively impacted by the combination of "blockwise transfer" with RDC.

Yet for all simulations CoAP outperforms HTTP by a broad margin in respect to the request latency. On the other hand the numbers indicate that CoAP is **very sensitive to payload increase**, in particular if it results in a "blockwise transfer" and in combination with RDC. This is very much comprehensible, considering that CoAP draws its main advantage from reducing overhead to an absolutely necessary minimum in terms of payload and even more so in terms of amount of exchanged individual packets, which will be significantly change with increased payload sizes and the resulting increased number of transmitted packets, where on the other hand the effects of RDC on the latency will be multiplied.

Dropped Messages For the aforementioned latency measurements, the response timeout of 10 seconds had to be relaxed for some configurations, so that comparable numbers could be gathered and also because remnants of old requests circulating through the network would impact the following requests. Under the assumptions made and described in Section 6.1.2, the responses exceeding this timeout count as "dropped messages". Table 6.2 illustrates the respective numbers.

	2 requests	4 requests
64B	0%	9.2%
128B	0%	7.6%

(a) 1 hop

	2 requests	4 requests
64B	0.4%	49.6%
128B	2.8%	56.4%

(b) 2 hops

Table 6.2: "Dropped messages" for HTTP

CoAP did not encounter request latencies above the timeout interval of 10 seconds under the tested configurations, which is also the case for HTTP with only one request issued at a time. Only the numbers for the remaining simulation configurations are thus presented.

For two parallel requests issued, the unsuccessful requests are only significant for **128 bytes payload**. For four parallel requests, all configurations would yield "dropped messages", although the differences between the two payload sizes applied are insignificant as they are still inside the margin of deviation.

These results are coherent with the latency measurement results in Section 6.2.2.

6.3 CoAP

As follow up on the comparative analysis between HTTP and CoAP (see Section 6.2), the limitations and more detailed characteristics of CoAP are investigated, under similar assumptions and in respect to the same observed metrics.

6.3.1 Description

The following metrics are surveyed:

- **Latency** for various hop counts and payloads.
- **Energy consumption** for different hop counts and payloads.
- The **maximum hop** count possible with **128 bytes payload**.
- **Dropped messages** for various hop counts and **128 bytes payload**

Similar topologies to the ones used in Section 6.2 are used, but with only one participating server. The numbers of hops between router and server are varied, which is indicated by the dashed line between the gray nodes in Figure 6.11.

In difference to the simulations in Section 6.2, the **client** issuing the requests is also implemented as a simulated node *inside* the simulated network. This was desirable in the first place and now possible because no HTTP client is required. Additionally, it turns out to be necessary,

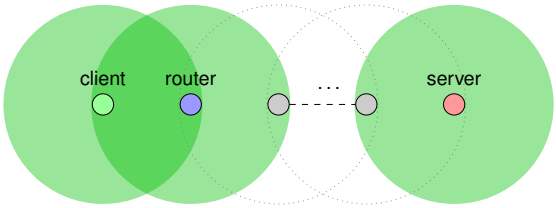


Figure 6.11: Simulation topology CoAP

because running the simulations at 100% speed cannot be guaranteed for all simulations with high hop counts, which would introduce adverse effects to the simulation, i.e.spurious client time-outs due to different clocks. Following this change, multiple clients cannot be synchronized in the manner of Section 6.2.2, thus no topologies with multiple requests are simulated. On the other hand, it is now possible to measure the latency of a request directly at the link between the client and following hop (i.e. the border-router) in contrast to the indirect method used in Section 6.2. Please note that this also means that the resulting numbers are not comparable to those of Section 6.2.

All resulting numbers are the average of 5 repeated runs. Both, client and server are based on the respective Erbium examples[13] and the same codebase of Contiki was used as in Section 6.2.

The *client* is either set to issue *continuous requests* for latency and power consumption measurements or *exactly 10 requests* for dropped messages measurements. For the "dropped messages" measurements the client is modified to include a "cool off" phase of 10 seconds after each request to minimize the effects on the subsequent requests, resulting in requests only being sent every 20 seconds instead.

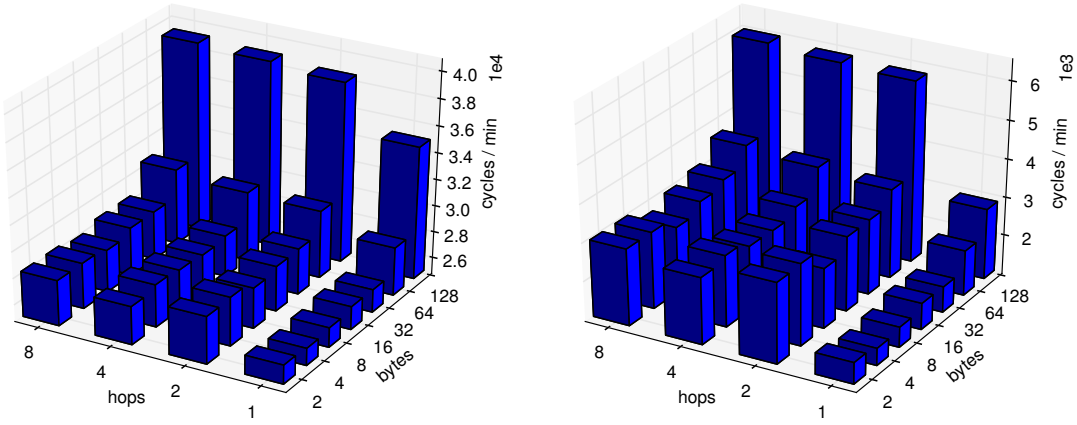
6.3.2 Results

Discussion Power Usage

Figures 6.12a, 6.12b and 6.13a visualize the collected numbers for the power usage for CoAP for different hop/payload combinations (see Section 6.1.2). Please note that the **lower limits** of the time-axis in the discussed bar-charts are adjusted for better visibility of the changes in value.

CPU active time For payloads between 2 and 32 bytes and hop counts between 2 and 8, the CPU active time varies only marginally (see Fig. 6.12a). For hop count equals 1, the CPU active time is visibly less than for all higher hop counts, which might be explained by the one single hop being the border-router, which does not use RDC (see Fig. 6.11). It was already assumed in Section 6.2, that an adjacent hop reduces the number of necessary transmissions by the server if RDC was disabled, as packet-trains are suppressed, which means the server node can return to power saving modes earlier than with higher hop counts. Apart from this step, **additional hops** apparently **introduce only marginal changes to power consumption** regarding the number of active CPU time.

[13] https://github.com/bipson/contiki-simfork/blob/c1b466a797/examples/iotsys-sim/er-obix-server.c and https://github.com/bipson/contiki-simfork/blob/c1b466a797/examples/iotsys-sim/er-obix-requester.c



(a) CPU active time

(b) Radio transmit time

Figure 6.12: Cumulative numbers CoAP - RDC enabled

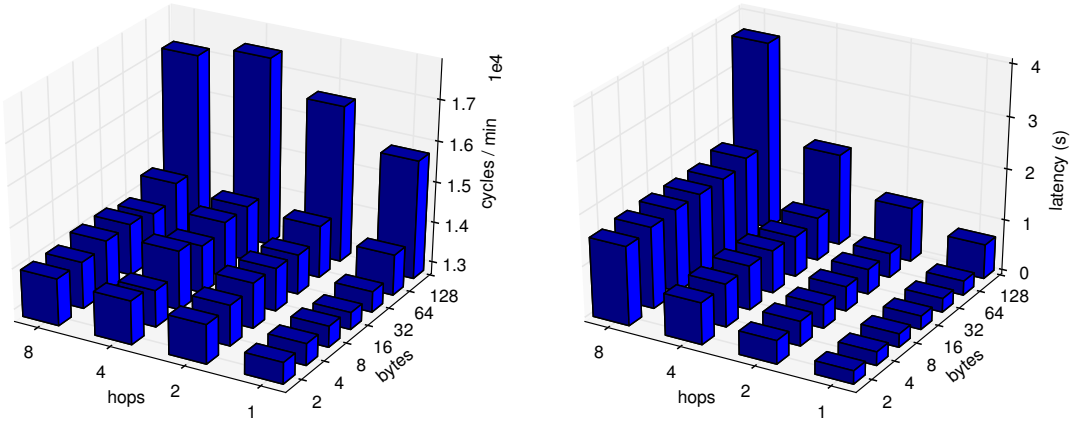
For the scenarios with a payload equal to 64 bytes, a slight bump can be recognized compared to the lower payloads, which reflects that **IEEE 802.15.4 fragmentation** was necessary to send the payload. A payload of **128 bytes** does further introduce a "blockwise transfer" by CoAP (see Section 2.10.1), which causes a *considerable* increase in CPU active time.

Radio transmit time The numbers for time spent actively transmitting shows very much the same effects as the numbers for the CPU active time, with a distinct impact of the RDC-disabled single hop in the 1 hop scenario, which is coherent with the previous results (see Fig. 6.12b). Compared to CPU active time the impact is much bigger, so that the 128 bytes and 1 hop combination spends roughly as much energy for radio transmission as the combinations with more hops do for as little as 2 bytes. Although this would indicate a huge impact of RDC on energy consumption, the energy savings of RDC with time spent actively listening weigh considerably more (cf. Section 6.2.1).

Again, for hop counts from 2 to 8 and payloads between 2 and 32 bytes the values vary only slightly. Some variations, as seen for the 8 bytes and 2 hops combination, are visible, but these are deemed to be inside the fluctuation range. For reasons not completely understood, some simulation runs would yield consistently lower results, and these variations happened through all combinations, but not with the same commonness, which may be solved by executing additional iterations of the simulations, but would have been unreasonably time consuming.

The same increase in energy consumption for payloads of 64 bytes as for CPU active time is visible for radio transmit time, most certainly also due to IEEE 802.15.4 fragmentation.

Radio listen time Interestingly enough, as for the active CPU time and time spent actively transmitting, the numbers for 1 hop are significantly lower than for the other topologies, although the the reasons for a correlation between listening time of a node and the RDC settings of an adjacent node are not accessible to the author and have to be analyzed in more detail.



(a) Radio listen time

(b) Latency

Figure 6.13: Cumulative numbers CoAP - RDC enabled

As for active listening time, some slight fluctuations are visible, e.g. in the 8 bytes and 4 hops combination, which are as well inside the fluctuation range (see Fig. 6.13a).

Again a significant "bump" can be spotted for payloads of 64 bytes, which is again most certainly caused by the necessary IEEE 802.15.4 fragmentation, although the impact is significantly lower than for active CPU time and also seems slightly lower than for actively transmitting time. The impact of "blockwise transfer" is very much comparable to the one on CPU active time, but seems to weigh heavier than IEEE 802.15.4 fragmentation.

Discussion Latency

In Figure 6.13b we can see the visualization for the impact of the different variables on the latency[14].

As expected, the hop count has a significant impact on the latency, with approximately linear growth. Please note that the simulated hop counts grow with a granularity of 2^n.

It is clearly visible, that in difference to the power consumption, the latency is only slightly impacted by IEEE 802.15.4 fragmentation at payloads equal 64 bytes, as the fragmentation takes place between adjacent nodes, while the fragmentation caused by the "blockwise transfer" mechanism of CoAP is easy to spot, approximately **doubling the latency**, which can be explained by the client-to-server fragmentation of "blockwise transfer". Overall, "blockwise transfer" seems to cause adverse effects, as application layer fragmentation will be happening between the endpoints of the transaction, while IEEE 802.15.4 fragmentation would happen between every intermediate hop, which might reduce energy consumption and latency of the overall transaction for larger hop counts and thus could be preferable, if there is a choice between both.

[14]Please note that a slightly different setup was used than in Section 6.2.2, results are thus not to be compared directly.

Drop Count & Maximum Hops

It was hardly possible to establish a connection with a CoAP server that was further away than **14 hops** of the client when RDC was enabled. The messages were not relayed correctly by the intermediate hops, which indicates a routing failure of some kind, although the RPL-implementation in Contiki allegedly works for far larger hop numbers [112]. The exact reasons have to be analyzed in more detail in future examinations. Very similar routing errors could rarely also appear at hop counts smaller or equal 14, but never commonly enough to reproduce.

For hop counts smaller or equal 14, the numbers depicted in Figure 6.14 were gathered. The rate of messages not (completely) received in a time frame of 10 seconds grows linearly up to **40% at 14 hops** and is as low as **2% for 10 hops**. It is assumed by the author, that most WSANs wont have hop counts anywhere near these numbers and it is also necessary to state that the "dropped messages" were deliberately caused by introducing a response timeout of 10 seconds and using a response payload of 128 bytes payload, which causes long-going "blockwise transfers" in CoAP.

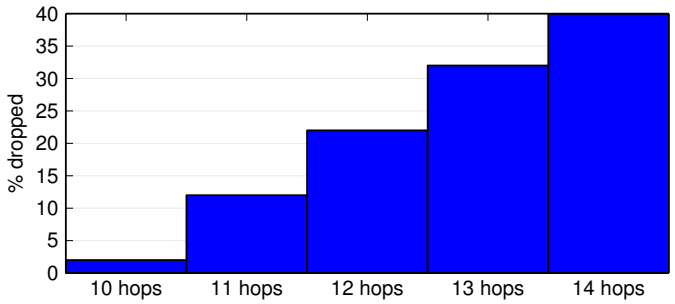


Figure 6.14: Dropped messages for CoAP - RDC enabled, 128B payload, 10s timeout

6.4 CoAP Group Communication

6.4.1 Multicast versus Unicast

Description

To measure the efficiency of group communication for CoAP, the setups in Figure 6.15 are simulated. Figure 6.15a shows a simple "blossom"[15] where a client would send a PUT-request to 4 individual servers. The servers were placed so that they were not visible to each other, in terms of directly reachable via radio.

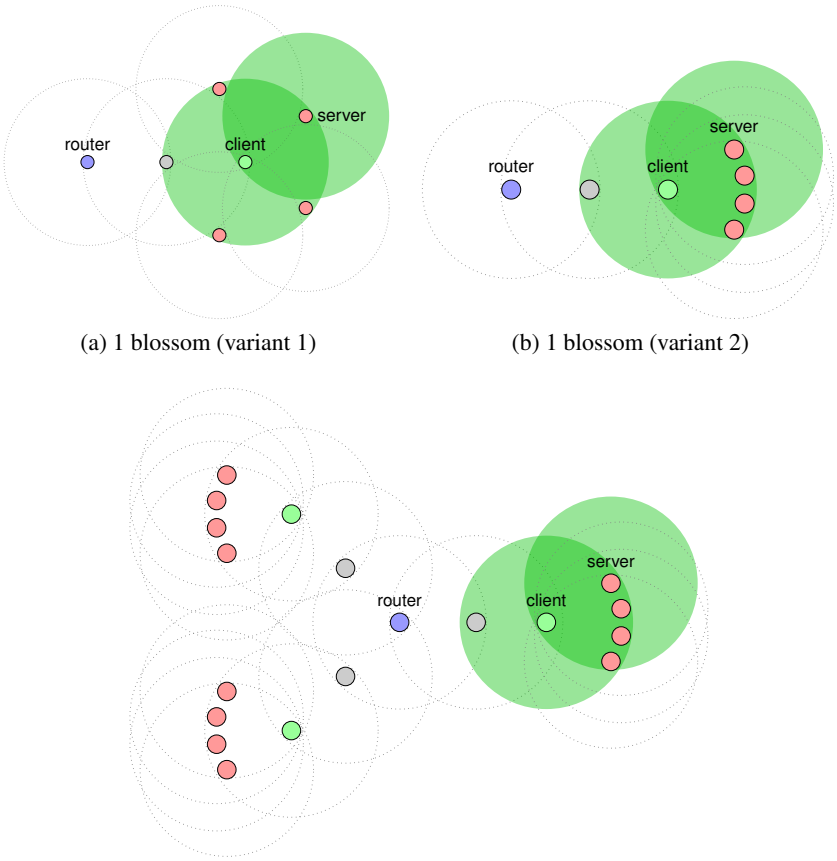
The first variation of this setup is the one in Figure 6.15b, where a blossom is completely interconnected, so that every node in the blossom is visible to the other nodes of the same blossom, under the assumption that interconnections between nodes will commonly occur in

[15] A "blossom" describes a geographical grouping of devices in the topology which emerges from the otherwise tree-like structure, in this case consisting of a client and four nearby servers.

real environments, and to determine the impact of this change compared to the aforementioned setup.

To test for scalability performance, yet another variation seen in Figure 6.15c is implemented, which is composed of three fully interconnected blossoms, with every client sending periodic requests to the servers in the respective blossom only.

The payload sent with each PUT-request is **16 bytes** and the results are taken from the averages of **5 repeated simulations**.



(a) 1 blossom (variant 1)

(b) 1 blossom (variant 2)

(c) 3 blossoms

Figure 6.15: Simulation topologies CoAP nulticast vs. unicast

These setups were run in two different modes:

- **Unicast** This corresponds to the state of Erbium at the time of writing, i.e. the client would send an **individual**, **blocking request** with payload for every queried server, thus sending sequential queries, where each of of them would wait for the respective acknowledgment[16].

[16]Unfortunately, the Erbium implementation would not successfully work with "non-confirmable" requests under this scenario, and attempts to implement these were unsuccessful and abandoned after some time.

- **Multicast** The client sends a transient link-local multicast datagram (containing the non-confirmed request and payload). The servers inside that blossom are configured to listen for messages sent to said multicast address and handle the respective CoAP request. Please note that Contiki had to be adapted to make multicasting work[17].

The application code for client and server were based on Erbium examples and adjusted accordingly, where for the multicast examples, group communication infrastructure had to be added to Erbium[18] (see Section 5.1.3).

Power consumption The energy consumption of all **server** nodes, the hop and the **client** node in the observed blossom is shown, according to Section 6.1.2. For the server nodes, the average of all 4 nodes is shown.

Latency The request latency was measured at the **client** and **servers**, i.e. the time difference between the first request of this interval being sent by the client, to the last server receiving its request.

Results

Power consumption Figure 6.16 shows the impact of multicasting for the different participants in the WSAN, on **CPU active time**, **active radio transmit time** and **active radio listening time** for each topology (see Section 6.1.2 for a detailled description of these metrics). The numbers are shown as percentage increase over the equivalent unicast results.

Interconnected blossoms Variant 1 and 2 of the single blossom setup seem to result in slight differences when comparing multicasting with unicasting, but a general trend is difficult to discern and the results are considered to be subject to random effects, as the fully interconnected nodes in the blossom can result in various individual manifestations of the underlying DODAGs, some of which will perform better than the others.

Hop To measure the impact on nodes that are not active participants in the transaction itself, by being neither sender, receiver or relay, the same numbers were also collected for the hop node, which is only connected to the client of the respective blossom (see Fig. 6.15). This node is presumably untouched by unicast transactions between a client and the adjacent servers, thus the vast increase in power consumption for the hop node supports the assumptions made about flooding being the root cause for increased power consumption at other nodes.

[17] At the time of writing, Contiki would not support multicast routing, which led to the implementation of the LOWPAN_BC0 multicast mechanism. The respective changes are described in Section 5.1.4.

[18] https://github.com/bipson/contiki-simfork/blob/e5d1236df0/examples/iotsys-sim/er-obix-gpost-server.c, https://github.com/bipson/contiki-simfork/blob/e5d1236df0/examples/iotsys-sim/er-obix-group-uc-sender.c and https://github.com/bipson/contiki-simfork/blob/e5d1236df0/examples/iotsys-sim/er-obix-group-mc-sender.c

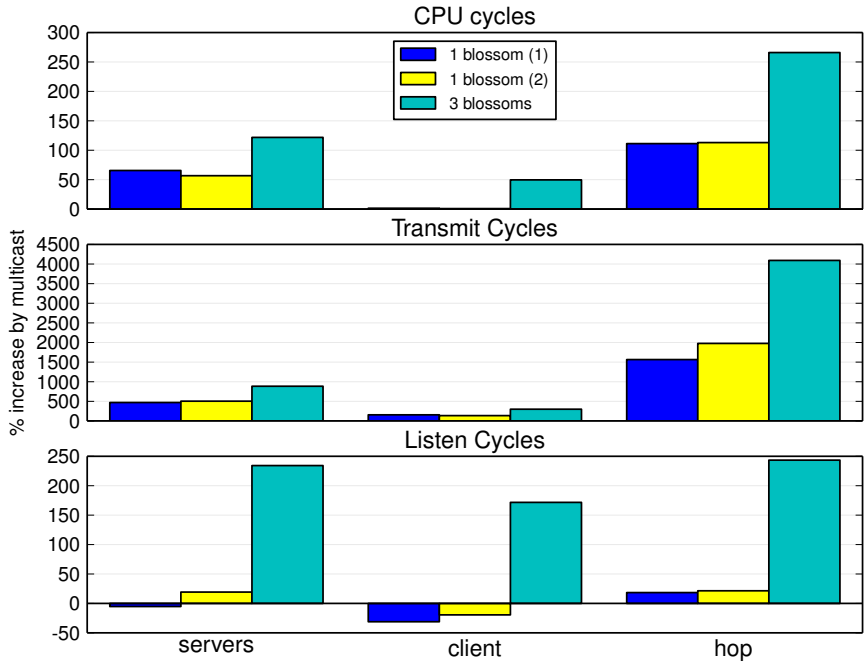


Figure 6.16: Power consumption for multicast vs. unicast at 16 bytes payload

Additional Blossoms The assumption about multicast messages flooding the network is furthermore supported by the significant increase between the two single blossom setups and the three blossom setup, where three clients are expected to flood the network instead of just one, resulting in significantly higher power usage for all observed nodes. The effects of the 3 blossom setup are thus ignored in the following paragraphs.

CPU cycles While for the servers the used multicasting mechanism results in a significant increase in CPU active time (**65.6%/56,8%**), the increase for the client is only **1.5%** or **0.8%** for the single blossom setup, which is considered to be an insignificant change.

Transmit cycles Compared to unicasting, multicasting results in a vast increase of cycles spent with the radio actively transmitting, with the client having the most modest increase of all nodes. For the client the increase is **136.36%**, respectively **156.56%** for the two single blossom setups.

Listen cycles In difference to the previously stated numbers, the numbers for cycles spent actively listening for multicasting are comparable to the numbers for unicasting. For the client they even are marginally *lower* which indicates an reduced energy consumption, partially due

to the absent acknowledgments for multicasting, which effectively reduce the number of radio transmissions in the clients vicinity.

Overall A negative impact of multicasting on power consumption is visible for almost all observed numbers, in particular for the three blossoms setup. This can be explained by the respective messages **flooding** the network and thus being processed by the nodes more often than unicast messages. For the other setups, the increase is tangible, but most dramatic with the numbers for **radio transmitting**. The room for improvements regarding the energy consumption caused by LOWPAN_BC0 is apparent.

Latency In Figure 6.17 the latencies measured for unicast and the equivalent multicast setups are compared.

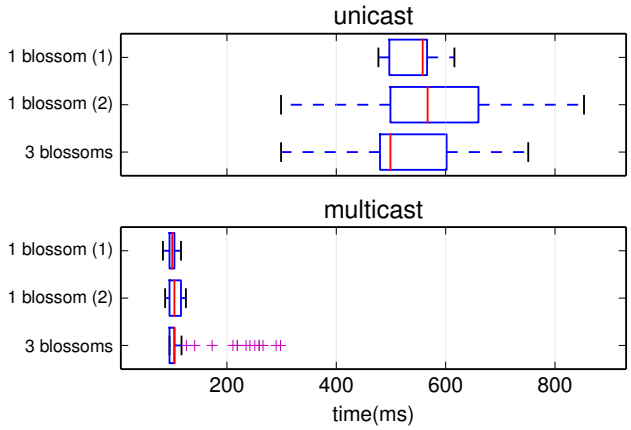


Figure 6.17: Latency for multicast vs unicast at 16 bytes payload

The significantly better performance of multicasting is *apparent* and easy to explain, considering the fact that unicasting results in **sequential requests**, where the servers will be queried one at a time and the confirmation for each server awaited until the next server is queried. An increase in outliers for multicasting is visible for the 3 blossom setup, which might be attributed to the flooding effects of multicasting. As only four group members are modeled in the simulation, this advantage of multicasting is expected to grow dramatically with increased group sizes, but be also more impacted by an increased number of groups and by higher frequency of transmissions.

6.4.2 Unicast vs. LOWPAN_BC0 vs. SMRF vs. MPL

The LOWPAN_BC0 multicast mechanism used in Section 6.4.1 is arguably straightforward in terms of design and resembles broadcasting e.g. it does not use routing information typically used for IPv6 multicasting, which might or might not be available in a LLN, depending on the routing infrastructure. Contiki incorporates two built-in multicast engines, SMRF and MPL,

which are considered to be advanced alternatives for multicasting in WSANs (see Sections 2.8.1 and 2.8.2). The following round of simulations will compare the LOWPAN_BC0 mechanism to those multicast engines.

Description

Figure 6.18 shows the topology to be simulated. In difference to the simulations in Section 6.4.1, a hop is introduced between the client and the servers, because no multicast routing is involved without at least one hop between multicast datagram emitter and receivers, and thus no multicast routing engines would be required at all.

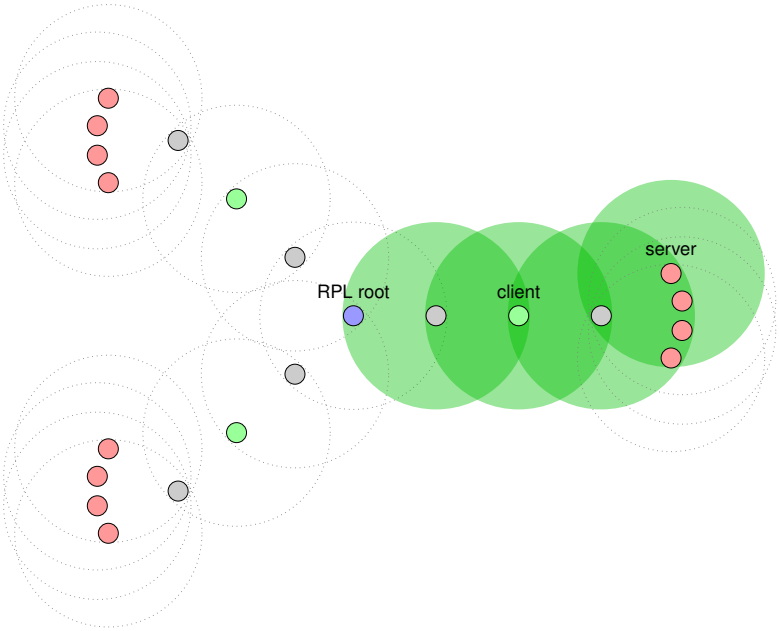


Figure 6.18: Simulation topology multicast engines

The topology consists of **three blossoms**, each one forming a communication group, where the client would send requests to the four server inside the same blossom.

Each simulation consists of consecutive `POST`-requests at a **10 seconds interval** and **32 bytes payload**. Every simulation is repeated 5 times and the results are averaged over all runs.

Latency numbers are gathered by measuring the time difference of the timestamps of mote console outputs[19] between:

1. the client stating that the requests are prepared and will be sent
2. the last server of that client stating that the request was received, processed and the payload correct and complete

Only the nodes from one blossom are observed.

[19]Please note that in the foregoing simulations the radio logs were used instead of the mote console output

Every server not stating reception of a complete and correct datagram inside the request interval of 10 seconds would result in a **"dropped message"**, *regardless* of the blossom the server belongs to. The numbers are then normalized as percentage of all expected packets.

Power consumption is measured in one single blossom at the **client**, the **two hops** and the **four servers**. For the hops and the servers the average of all affected nodes are calculated.

Further RDC is not used for this round of simulations, because the implementation of MPL would not fit on a node running Erbium aside ContikiMAC. Additional insights on the memory footprint of MPL were already discussed in Section 5.2.2.

As for Section 6.4.1, the unicast example would use *Confirmable* requests, which would mean that an additional acknowledgment would be sent as reply for every request and server, as the Erbium implementation would not successfully work with *Non-confirmable* requests under this scenario, and attempts to implement these were unsuccessful and abandoned after some time.

The multicast engines SMRF and MPL can, and *should* be parameterized, to optimize for certain environments [27]. The SMRF implementation in Contiki queries the lower layers for the optimal values, while for MPL I_{min} was set to 125 ms, adhering to the implementers notes.

Contiki did not only gain support for the mentioned multicast routing mechanisms, but was also changed in other parts. Also Erbium underwent a major restructuring and was adjusted to the now finalized CoAP RFC[20] [6]. The applications are based on the ones used in Section 6.4.1 and adjusted accordingly[21].

Discussion Results

Latency The results for latency are depicted in Figure 6.19. Please note the **logarithmic scale**.

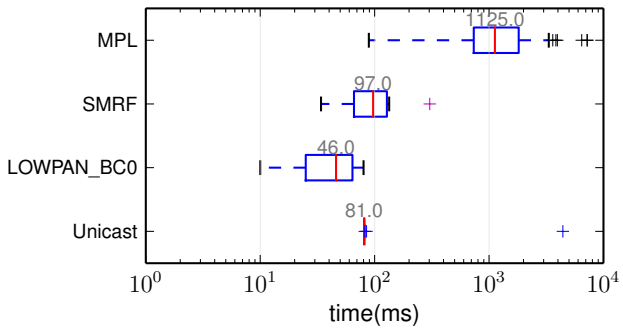


Figure 6.19: Latency comparisons group communication

Despite the logarithmic scale, a clear disadvantage for the MPL engine is distinguishable at first glance, which means that MPL yields a latency that is higher as all other simulated engines *by a factor of 10*, resulting in even unicasting yielding better results on average and only one

[20] https://github.com/contiki-os/contiki/tree/d7cf3b1f74

[21] https://github.com/bipson/contiki-simfork/tree/f50adaf2d2/examples/iotsys-sim

outlier reaching the realms of MPL[22]. Also remarkable is the distribution for MPL compared to the other engines, ranging from results below 100 ms, up to nearly 10 seconds.

The other simulated engines are not as easy to compare with a logarithmic scale, but still possible to say that the LOWPAN_BC0 mechanism on average yields significantly better results than all other engines, while unicasting and SMRF are very close with a significantly different IQR, meaning that, besides one arguably distant outlier, unicast yields very predictable results, where SMRF introduces some variation, most certainly caused by the forwarding delay of each node being calculated as random number. The distribution of results for the LOWPAN_BC0 mechanism can be attributed to the same reason, a random number being used for the forwarding delay, where both engines use different lower and upper bounds, which also indicates that both engines could yield very similar results if they were configured accordingly.

A difference between these results and the RDC-enabled results from Section 6.4.1 is discernible, which might be explained with RDC delaying the individual, sequential requests more, causing bigger latencies for unicasting.

Power consumption The power consumption caused by the different engines is shown in Figure 6.20. Please note that again the figure uses a **logarithmic scale**.

As for latency, the numbers for MPL are *far* from the numbers of all other engines. This is most certainly caused by a "storm" of ICMPv6 messages being constantly exchanged by the nodes, which was discovered during tests of MPL with this and other topologies. It appears that the used topology exploits this behavior, as it has a low density. MPL uses ICMPv6 messages to negotiate if new information must be shared with other nodes (see Section 2.8.2) and can be tweaked by adjusting the Trickle timers of MPL, which was not done besides using the default values as advised by the implementors of the engine. Although less afar than MPL, the numbers for unicast are still well above the numbers for LOWPAN_BC0 and SMRF, which is understandable, considering that for unicasting, each client would have to send four individual messages to every server instead of one datagram for all four. Additionally, due to only "confirmable" CoAP messages being sent for unicast, the power consumption of the acknowledgment contributes to the numbers, which unfortunately was not avoidable.

Dropped messages Figure 6.21 illustrates the percentages of messages that would *not* reach their intended recipient during the observed interval.

Both unicast and LOWPAN_BC0 experience comparable packet losses, for supposedly different reasons. For unicast, this is most definitely caused by the aforementioned RPL infrastructure faults/routing errors. These become apparent on examination of the log files corresponding with the simulations, where a request to the same node would often fail for several subsequent requests, and sometimes the whole duration of the simulation. Additionally, due to the implementation of the requests to multiple endpoints by one client as series of sequential *blocking*

[22]This unicast outlier is probably caused by a fault in the RPL infrastructure, which can result in routing errors and (timely restricted) unreachable nodes, which cause delayed responses if the node just becomes available during a resent request. The reasons to suspect RPL routing faults as cause will be revisited further down.

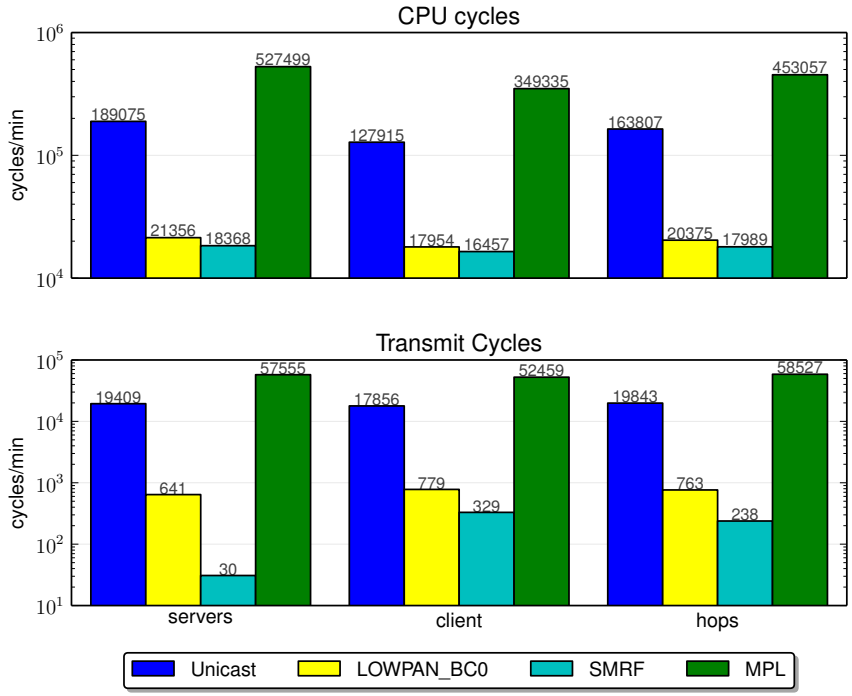


Figure 6.20: Power comparisons group communication

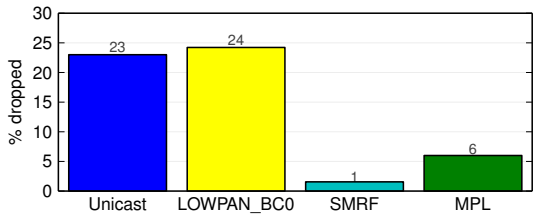


Figure 6.21: Dropped messages comparison group communication

requests[23], an unresponsive node for one server may cause up to 3 additional dropped packets that might otherwise be successful requests. In theory, with a fully working RPL infrastructure, unicast *should* result in zero dropped packets, particularly as acknowledged requests were used. The numbers for LOWPAN_BC0 on the other hand seem reasonable, but also depend on the used topology. Still the results show that there is room for improvement with the LOWPAN_BC0 mechanism.

The multicast engines SMRF and MPL are at the other end of the spectrum, yielding very small numbers for dropped packets. Contrary to the findings in [27], MPL experiences *slightly*

[23]The process responsible for the requests is suspended for every single request until the corresponding acknowledgment "ACK" is received or the timeout reached.

higher packet loss than SMRF for the simulated topology. It seems that MPL experiences a corner case in the simulated environment that would cause an undesirable behavior also encountered with the numbers for latency and energy consumption.

CHAPTER 7

Evaluation and Outlook

7.1 Discussion

The Internet has brought massive changes and a full set of new technologies to the world of software engineering, and Internet computing has become a separate discipline of distributed computing with its own set of challenges and technologies, affecting the daily life of many people. The IoT is bringing the Internet to a huge amount of previously stray devices, interconnecting them and making them (in principle) accessible to the users of the Internet and its devices. At the same time the IoT brings a new set of devices and corresponding challenges to the Internet and thus to the field of Internet computing. This will require new technologies and paradigms to be researched and developed by and for the future software engineers.

The hypotheses for this thesis, listed in Section 1.4, were:

Hypothesis *The suggested solution is to use Web technologies on top of a communication stack optimized for constrained devices which still enables integration of constrained devices into an Internet of Things. Such a solution should be* ***feasible*** *and offer an* ***equivalent feature set*** *to existing solutions, particularly regarding group communication.*

Hypothesis *The possibility for group communication and asynchronous communication allows a solution to* ***scale well*** *for various larger, fragmented and complex network structures and in particular for* ***diverse communication scenarios****.*

The first hypothesis could be confirmed. General feasibility of the protocol stack on constrained devices was shown and in comparison to HTTP, CoAP has proven to be *vastly* more efficient in terms of **request latency** and **energy consumption**, and better suited for constrained environments. Given the fact that CoAP might some day even *exceed* the feature set of HTTP to be even better adapted to the requirements of WSAN applications (see Section 2.10.2), it is the **preferable choice** for deployments of RESTful services in a WSAN.

Compared to other approaches trying to bring web services to the IoT, CoAP has the advantage of being an IETF Request for Comments (RFC), building on established experiences from the Web and the resulting paradigms. Its close affinity to HTTP and its REST characteristics will make future deployments of gateways easier. Also, CoAP enables a very lightweight operation, which sets it apart from most approaches presented in Section 3.1.

Regarding the second hypothesis, some open questions remain, as the simulated scenarios did not reach the intended scale. **Group communication** for CoAP is still to be agreed upon (see Section 2.10.2), but results of this work indicate that it is not only **feasible** and **sustainable**, but a **necessity** for certain scenarios that are expected to occur in WSANs, reducing the energy consumption and complexity of the participating devices and all other devices in the network. The shown implementation of a group communication infrastructure for CoAP is **efficient** and has a **small binary footprint**. Additionally, **multicast routing** in WSANs has shown to be a core enabler technology for group communication and **affects the efficiency of group communication *drastically***. The gathered results indicate that the current state of multicasting in RPL networks is not to be considered satisfactory.

Another issue which remains for service endpoints in constrained environments, is the resulting **payload size**, not only for message transportation and the resulting energy consumption, but most of all because of the **limited buffers** available on constrained devices. This issue is closely related to the **expensive serialization and de-serialization** of most conventional message encodings, e.g. XML. While in the past this was usually solved through legacy protocols with a binary encoding, this approach limits the ability to form heterogeneous networks without the use of respective gateways, which pollutes the basic idea of a distributed services interoperability based on open standards. The presented stack, based on OBIX with a **compact message schema** (cf. Section 4.1) and further EXI encoding, promises to solve this problem sufficiently. The gathered insights on the impact of different payload sizes on the devices, confirm the demand for this or a comparable solution. Nevertheless, implementation of an EXI parser for encoding and decoding of XML documents on the constrained device itself has shown to be difficult and several efforts were made to solve this problem (see Section 3.2). Although native EXI stream parsing on the device, avoiding XML intermediates completely, was not shown in this work, it is assumed to bring a sustainable solution to this dilemma.

7.2 Outlook

Simulations at a **larger scale** would be a desirable and possible continuation to this work, with a multitude of participating nodes and with more variations to the underlying topologies. Also, the surveyed **communication patterns** might be extended, to include CoAP *observe* (see Section 2.10.2) and other random scenarios into the comparisons. This would be feasible, if the framework for the simulations was overall more efficient at running the simulations itself and also more flexible as the proposed approach, which arguably has a makeshift characteristics and does not scale for larger node counts, larger numbers of simulations and topology variations. Especially larger topologies or variations that lead to changes in application code are unreasonably cumbersome, when the proposed framework is used. This could also be achieved by the use of dedicated **network simulators**, e.g. OMNeT++ [113], which promises quicker results, a more

established toolset and additional insights into the low-level metrics of the simulated scenario. It is expected though, that modeling of a complete 6LoWPAN&RDC/CoAP stack and its fine details in OMNeT++ or similar simulators is a very ambitious task, especially as long as some details are not yet finalized, e.g. the CoAP *observe* functionality. That being said, as this work has shown, every large scale simulation should anyway succeed a general feasibility process, e.g. using simulated deployment in Contiki, as the constraints on the devices are pressing.

One aspect of high significance for WSANs which was also not considered is the **mobility**[1] **of devices in RPL networks**, and the resulting changes of underlying routing structures. The effects of these changes on application layers and communication patterns, in particular in the context of unreliable multicast routing, are highly interesting, as these disruptions are very likely to happen in real world environments and robustness against them is desirable for administration.

Closely related to the aforementioned problem is the problem of multiple RPL DODAG instances and wireless devices moving between them, thus changing their IPv6 address through stateless address autoconfiguration. As the IPv6 address in a truly IPv6 based IoT is the equivalent to a unique identifier of a device, this renders the device unusable in a distributed context, if additional technologies for dynamic address resolution are not used. Although defining a Personal Area Network Identifier (PAN ID) per DODAG instance a respective device can be hindered from joining any other instances, this would also drastically reduce the ability of a WSAN infrastructure to repair itself through alternative routes, e.g. another DODAG instance if necessary. To the knowledge of the author, the technologies used in this work, together with the assumptions made, do not offer any immanent solutions to this problem, thus further research in this area would be highly recommended.

Multicasting for LLNs is still a very promising field for future developments and improvements. Certain assumptions made, e.g. only routing *downwards* a RPL tree by SMRF, should be discussed extensively, and thorough comparisons of different proposed routing algorithms should be executed in **various** environments and with a vast number of underlying topologies.

Another largely neglected field is the reliability of multicasting in LLNs. Depending on the requirements, certain predicted uses for group communication based multicasting will not be reasonable, if multicasting is accepted to be unreliable to a certain degree. Either group communication and its use cases must be defined in a more detailed way, so that a distinction can be made between certain types of group communication, or the efforts to define according multicast routing algorithms must consider this problematic.

Another field for further investigations are the **security** aspects of 6LoWPAN and CoAP and the effect of the respectively targeted methods on the proposed stack and the underlying network and devices. Security has yet a rather small importance in the involved projects and their communities, based on the academic character of the field and the currently immediate problems to be solved. Yet the anticipated future applications for the used technologies indicate that security aspects will one day play an immense role for their applicability, in the worst case rendering them useless if security is not available. Not only should the targeted security precautions be feasible, which is assumed, but their impact on other components, which already

[1] *Mobility* in this case can also mean a device, or any intermediate device in the routing structure, being unreachable e.g. because of interferences of any kind (e.g. *radio jam*, absorption, etc.) or having depleted its energy supply.

were or are currently being defined without these security measures activated, could in the worst case make past findings and the resulting decisions obsolete or erroneous.

APPENDIX A

Code Listings

A.1 Group Communication/Multicast changes to Erbium and Contiki

Listing A.1: "Diff" for CoAP group-communication changes

```
diff --git a/apps/er-coap-13/er-coap-13-engine.c
    b/apps/er-coap-13/er-coap-13-engine.c
index 9bde8b7..7b0b32e 100644
--- a/apps/er-coap-13/er-coap-13-engine.c
+++ b/apps/er-coap-13/er-coap-13-engine.c
@@ -71,6 +71,7 @@ PROCESS(coap_receiver, "CoAP Receiver");
 /*- Variables
     -----------------------------------------------------------------*/
 /*---------------------------------------------------------------------------*/
 static service_callback_t service_cbk = NULL;
+static group_comm_callback_t group_comm_cbk = NULL;
 /*---------------------------------------------------------------------------*/
 /*---------------------------------------------------------------------------*/
 /*---------------------------------------------------------------------------*/
@@ -97,6 +98,14 @@ coap_receive(void)

     coap_error_code = coap_parse_message(message, uip_appdata,
         uip_datalen());

+    if( uip_is_addr_mcast(&UIP_IP_BUF->destipaddr) && ((uint8_t
    *)(&UIP_IP_BUF->destipaddr))[1] >> 4 ){ // multicast address with
    transient scope
+       if(group_comm_cbk != NULL){
+               group_comm_cbk(&UIP_IP_BUF->srcipaddr,
    &UIP_IP_BUF->destipaddr, message->payload, message->payload_len);
+       }
+       PRINTF("#### RETURNING.\n");
+       return NO_ERROR;
```

```
+    }
+
     if (coap_error_code==NO_ERROR)
     {

@@ -302,6 +311,14 @@ coap_set_service_callback(service_callback_t callback)
 {
   service_cbk = callback;
 }
+
+/*-----------------------------------------------------------------------------*/
+void
+coap_set_group_comm_callback(group_comm_callback_t callback)
+{
+  group_comm_cbk = callback;
+}
+
 /*-----------------------------------------------------------------------------*/
 rest_resource_flags_t
 coap_get_rest_method(void *packet)
@@ -591,6 +608,14 @@ PT_THREAD(coap_blocking_request(struct request_state_t
    *state, process_event_t e

   PT_END(&state->pt);
 }
+
+void coap_simple_request(uip_ipaddr_t *addr, uint16_t port, coap_packet_t
+   *request){
+       uint8_t packet[COAP_MAX_PACKET_SIZE+1];
+       uint16_t packet_len;
+       packet_len = coap_serialize_message(request, &packet );
+       PRINTF("packet_len is %d.\n", packet_len);
+       coap_send_message(addr, uip_htons(port), &packet, packet_len);
+}
 /*-----------------------------------------------------------------------------*/
 /*- Engine Interface
    ------------------------------------------------------------*/
 /*-----------------------------------------------------------------------------*/
@@ -599,6 +624,7 @@ const struct rest_implementation
    coap_rest_implementation = {

   coap_receiver_init,
   coap_set_service_callback,
+  coap_set_group_comm_callback,

   coap_get_header_uri_path,
   coap_set_header_uri_path,
diff --git a/apps/er-coap-13/er-coap-13-engine.h
   b/apps/er-coap-13/er-coap-13-engine.h
index 4730a07..b6111ce 100644
--- a/apps/er-coap-13/er-coap-13-engine.h
+++ b/apps/er-coap-13/er-coap-13-engine.h
@@ -91,4 +91,6 @@ PT_THREAD(coap_blocking_request(struct request_state_t
```

```
    *state, process_event_t e
 }
 /*---------------------------------------------------------------------------*/

+void coap_simple_request(uip_ipaddr_t *addr, uint16_t port, coap_packet_t
    *request);
+
 #endif /* COAP_SERVER_H_ */
diff --git a/apps/erbium/erbium.h b/apps/erbium/erbium.h
index 0fecd96..97869c1 100644
--- a/apps/erbium/erbium.h
+++ b/apps/erbium/erbium.h
@@ -43,13 +43,14 @@
 #include <stdio.h>
 #include "contiki.h"
 #include "contiki-lib.h"
+#include "uip.h"

 /*
  * The maximum buffer size that is provided for resource responses and must
      be respected due to the limited IP buffer.
  * Larger data must be handled by the resource and will be sent chunk-wise
      through a TCP stream or CoAP blocks.
  */
 #ifndef REST_MAX_CHUNK_SIZE
-#define REST_MAX_CHUNK_SIZE     128
+#define REST_MAX_CHUNK_SIZE     64
 #endif

 #ifndef MIN
@@ -81,6 +82,12 @@ typedef void (*restful_response_handler) (void *data,
    void* response);
 /* Signature of the rest-engine service function. */
 typedef int (* service_callback_t)(void *request, void *response, uint8_t
    *buffer, uint16_t preferred_size, int32_t *offset);

+/* Group comm callback */
+typedef void (* group_comm_callback_t) (uip_ipaddr_t *sender_addr,
+        uip_ipaddr_t *receiver_addr,
+        uint8_t *data,
+        uint16_t datalen);
+
 /**
  * The structure of a MAC protocol driver in Contiki.
  */
@@ -168,6 +175,7 @@ struct rest_implementation {

   /** Register the RESTful service callback at implementation */
   void (* set_service_callback)(service_callback_t callback);
+  void (* set_group_comm_callback)(group_comm_callback_t callback);

   /** Get request URI path */
   int (* get_url)(void *request, const char **url);
```

Listing A.2: “Diff” for Contiki LOWPAN_BC0 changes

```
diff --git a/core/net/sicslowpan.c b/core/net/sicslowpan.c
index 844064c..bdf6fd0 100644
--- a/core/net/sicslowpan.c
+++ b/core/net/sicslowpan.c
@@ -66,12 +66,13 @@
 #include "net/rime.h"
 #include "net/sicslowpan.h"
 #include "net/netstack.h"
+#include "sys/ctimer.h"

 #if UIP_CONF_IPV6

@@ -264,6 +266,88 @@ static int last_rssi;
 /*--------------------------------------------------------------------*/
 static struct rime_sniffer *callback = NULL;

+/*--------------------------------------------------------------------*/
+/* LOWPAN_BC0 structures
    */
+/*--------------------------------------------------------------------*/
+
+
+// buffer of latest sequence nrs using LRU policy for replacement
+static struct ipv6_sequence sequenceBuffer[LOWPAN_BC0_SEQUENCE_BUF_LEN] =
    {{0}};
+
+// current index
+static int cur_ipv6_sequence = 0;
+
+void debug_ipv6_sequence_buffer(){
+       uip_ip6addr_t unspecified;
+       int i;
+       PRINTF("SEQUENCE BUFFER IS:\n");
+       uip_create_unspecified(&unspecified);
+       for(i = 0; i < LOWPAN_BC0_SEQUENCE_BUF_LEN; i++){
+               PRINTF("  ");
+               if(uip_ip6addr_cmp(&unspecified,&sequenceBuffer[i].srcip)){
+                       PRINTF("Unspecified.\n");
+               }
+               else{
+                       PRINT6ADDR(&sequenceBuffer[i].srcip);
+                       PRINTF(": %d\n", sequenceBuffer[i].sequence);
+               }
+       }
+}
+
+// sequence number of this node
+static uint8_t lowpan_bc0_sequence = 0;
+
+int isSequenceNewAndStore(uip_ip6addr_t *srcip, uint8_t sequence){
+       uip_ip6addr_t unspecified;
+       int i;
```

```
+
+       // check if sequence number for source ip has been recorded
+       uip_create_unspecified(&unspecified);
+       for(i = 0; i < LOWPAN_BC0_SEQUENCE_BUF_LEN; i++){
+               PRINTF("  ");
+               if(uip_ip6addr_cmp(srcip,&sequenceBuffer[i].srcip)){
+                       // found src ip
+                       PRINTF("Source IP: ");
+                       PRINT6ADDR(srcip);
+                       PRINTF("\nFound sequence number for src ip %d,
    packet sequence is: %d\n ", sequenceBuffer[i].sequence, sequence);
+
+                       // in general we want to keep the sequence number
    increasing but after 256 frames it is reset to 0
+                       // therefore the second clause assumes that the
    current sequence number including the last 10
+                       // numbers are old packets. if the sequence number
    is lower below this treshold it is assumed to
+                       // be a new packet.
+                       if(sequenceBuffer[i].sequence == sequence &&
    !(sequence < sequenceBuffer[i].sequence - 10)){
+                               // this is an old packet
+                               return 0;
+                       }
+                       else{
+                               // this is a new packet
+                               sequenceBuffer[i].sequence = sequence;
+                               return 1;
+                       }
+               }
+       }
+
+       // if there was no sequence number found create new entry
+       if(cur_ipv6_sequence == LOWPAN_BC0_SEQUENCE_BUF_LEN){
+               // end of array reached reset index and overwrite existing
    entries
+               cur_ipv6_sequence = 0;
+       }
+       PRINTF("\nCreating new sequence number entry for IPv6 address.\n");
+       uip_ip6addr_copy(&sequenceBuffer[cur_ipv6_sequence].srcip , srcip);
+       sequenceBuffer[cur_ipv6_sequence].sequence = sequence;
+       cur_ipv6_sequence++;
+       // sequence was new
+       return 1;
+}
+
+static struct ctimer retransmit_timer;
+
+static void retransmit_callback(void *ptr){
+       PRINTF("Retransmit callback called.\n");
+       send_packet(&rimeaddr_null);
+}
+
```

```
+
 void
 rime_sniffer_add(struct rime_sniffer *s)
 {
@@ -1295,7 +1377,39 @@ compress_hdr_ipv6(rimeaddr_t *rime_destaddr)
   uncomp_hdr_len += UIP_IPH_LEN;
   return;
 }
-/** @} */
+
+/*-------------------------------------------------------------------*/
+/** \name LOWPAN_BC0 header compression
+ * @{                                                               */
+/*-------------------------------------------------------------------*/
+/* \brief LOWPAN_BC0 header for multicasting
+ *
+ * For flooding based multicasting a simple 8 bit sequence
+ * number is used to avoid re-transmission of an already seen
+ * multicast packet.
+ *
+ * \verbatim
+ * 0               1                   2                   3
+ * 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+ * +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+ * | LOWPAN_BC0    | SEQUENCE NR   |IPv6 header and payload ...
+ * +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
+ * \endverbatim
+ */
+static void
+compress_hdr_bc0(rimeaddr_t *rime_destaddr)
+{
+  *rime_ptr = SICSLOWPAN_DISPATCH_BC0;
+  // remember my own broadcast sequence number
+  isSequenceNewAndStore(&UIP_IP_BUF->srcipaddr, lowpan_bc0_sequence);
+  *(rime_ptr + 1) = lowpan_bc0_sequence++; // sequence number
+  rime_hdr_len += SICSLOWPAN_BC0_HDR_LEN;
+  memcpy(rime_ptr + rime_hdr_len, UIP_IP_BUF, UIP_IPH_LEN);
+  rime_hdr_len += UIP_IPH_LEN;
+  uncomp_hdr_len += UIP_IPH_LEN;
+  return;
+}
+

 /*-------------------------------------------------------------------*/
 /** \name Input/output functions common to all compression schemes
@@ -1360,6 +1474,7 @@ send_packet(rimeaddr_t *dest)
 static uint8_t
 output(const uip_lladdr_t *localdest)
 {
+  PRINTF("IN SICSLOWPAN OUTPUT.\n");
   int framer_hdrlen;

   /* The MAC address of the destination of the packet */
```

```
@@ -1406,6 +1521,7 @@ output(const uip_lladdr_t *localdest)
     * broadcast packet.
     */
    if(localdest == NULL) {
+         PRINTF("SENDING BROADCAST FRAME.\n");
      rimeaddr_copy(&dest, &rimeaddr_null);
    } else {
      rimeaddr_copy(&dest, (const rimeaddr_t *)localdest);
@@ -1415,21 +1531,44 @@ output(const uip_lladdr_t *localdest)

    if(uip_len >= COMPRESSION_THRESHOLD) {
      /* Try to compress the headers */
-#if SICSLOWPAN_COMPRESSION == SICSLOWPAN_COMPRESSION_HC1
-    compress_hdr_hc1(&dest);
-#endif /* SICSLOWPAN_COMPRESSION == SICSLOWPAN_COMPRESSION_HC1 */
-#if SICSLOWPAN_COMPRESSION == SICSLOWPAN_COMPRESSION_IPV6
-    compress_hdr_ipv6(&dest);
-#endif /* SICSLOWPAN_COMPRESSION == SICSLOWPAN_COMPRESSION_IPV6 */
-#if SICSLOWPAN_COMPRESSION == SICSLOWPAN_COMPRESSION_HC06
-    compress_hdr_hc06(&dest);
-#endif /* SICSLOWPAN_COMPRESSION == SICSLOWPAN_COMPRESSION_HC06 */
-  } else {
-    compress_hdr_ipv6(&dest);
+         if(localdest == NULL && ((uint8_t *)(&UIP_IP_BUF->destipaddr))[1]
    >> 4 ){ // broadcast packet --> use LOWPAN_BC0
+                 // check for transient ipv6 multicast address
+
+                 PRINTF("sicslowpan: Using BC0 compression.\n");
+                 PRINT6ADDR(&UIP_IP_BUF->destipaddr);
+                 compress_hdr_bc0(&dest);
+         }
+         else{
+               #if SICSLOWPAN_COMPRESSION == SICSLOWPAN_COMPRESSION_HC1
+                       PRINTF("sicslowpan: Using HC1 compression.\n");
+                       compress_hdr_hc1(&dest);
+               #endif /* SICSLOWPAN_COMPRESSION ==
    SICSLOWPAN_COMPRESSION_HC1 */
+               #if SICSLOWPAN_COMPRESSION == SICSLOWPAN_COMPRESSION_IPV6
+                       PRINTF("sicslowpan: Using IPv6 compression.\n");
+                       compress_hdr_ipv6(&dest);
+               #endif /* SICSLOWPAN_COMPRESSION ==
    SICSLOWPAN_COMPRESSION_IPV6 */
+               #if SICSLOWPAN_COMPRESSION == SICSLOWPAN_COMPRESSION_HC06
+                       PRINTF("sicslowpan: Using HC06.\n");
+                       compress_hdr_hc06(&dest);
+               #endif /* SICSLOWPAN_COMPRESSION ==
    SICSLOWPAN_COMPRESSION_HC06 */
+                 }
+         }
+  else
+  {
+         if(localdest == NULL && ((uint8_t *)(&UIP_IP_BUF->destipaddr))[1]
    >> 4 ){ // broadcast packet --> use LOWPAN_BC0 if transient ipv6
```

```
    multicast address
+                 PRINTF("sicslowpan: Using BC0 compression.");
+                 PRINT6ADDR(&UIP_IP_BUF->destipaddr);
+                 compress_hdr_bc0(&dest);
+         }
+         else{
+               PRINTF("sicslowpan: else Using IPv6 compression.\n");
+               compress_hdr_ipv6(&dest);
+         }
    }
+
    PRINTFO("sicslowpan output: header of len %d\n", rime_hdr_len);

-   /* Calculate NETSTACK_FRAMER's header length, that will be added in the
    NETSTACK_RDC.
+   /* Calculate  NETSTACK_FRAMER's header length, that will be added in the
    NETSTACK_RDC.
     * We calculate it here only to make a better decision of whether the
        outgoing packet
     * needs to be fragmented or not. */
 #define USE_FRAMER_HDRLEN 1
@@ -1451,7 +1590,9 @@ output(const uip_lladdr_t *localdest)
    framer_hdrlen = 21;
 #endif /* USE_FRAMER_HDRLEN */

-   if((int)uip_len - (int)uncomp_hdr_len > (int)MAC_MAX_PAYLOAD -
    framer_hdrlen - (int)rime_hdr_len) {
+ // if((int)uip_len - (int)uncomp_hdr_len > (int)MAC_MAX_PAYLOAD -
    framer_hdrlen - (int)rime_hdr_len) {
+   // framer_hdrlen --> should be already considered in MAC_MAX_PAYLOAD
+   if((int)uip_len - (int)uncomp_hdr_len > (int)MAC_MAX_PAYLOAD -
    (int)rime_hdr_len) {
 #if SICSLOWPAN_CONF_FRAG
     struct queuebuf *q;
     /*
@@ -1567,6 +1708,8 @@ output(const uip_lladdr_t *localdest)
     memcpy(rime_ptr + rime_hdr_len, (uint8_t *)UIP_IP_BUF + uncomp_hdr_len,
           uip_len - uncomp_hdr_len);
     packetbuf_set_datalen(uip_len - uncomp_hdr_len + rime_hdr_len);
+
+    PRINTF("# NOW SEND PACKET\n");
     send_packet(&dest);
   }
   return 1;
@@ -1588,6 +1731,7 @@ output(const uip_lladdr_t *localdest)
 static void
 input(void)
 {
+  uint16_t retransmission_wait_ms = 0;
   /* size of the IP packet (read from fragment) */
   uint16_t frag_size = 0;
   /* offset of the fragment in the IP packet */
@@ -1750,6 +1894,31 @@ input(void)
```

```
       rime_hdr_len += UIP_IPH_LEN;
       uncomp_hdr_len += UIP_IPH_LEN;
       break;
+    case SICSLOWPAN_DISPATCH_BC0:
+          PRINTFI("sicslowpan input: BC0\n");
+          rime_hdr_len += SICSLOWPAN_BC0_HDR_LEN;
+
+          /* Put uncompressed IP header in sicslowpan_buf. */
+          memcpy(SICSLOWPAN_IP_BUF, rime_ptr + rime_hdr_len, UIP_IPH_LEN);
+
+          // re-transmit packet
+          PRINTF("Check sequence number\n");
+
+          if(isSequenceNewAndStore(&SICSLOWPAN_IP_BUF->srcipaddr, (uint8_t)
    *(rime_ptr +1))){
+                //debug_ipv6_sequence_buffer();
+                retransmission_wait_ms = abs(random_rand() % 10); // max
    100 ms
+                //send_packet(&rimeaddr_null);
+                ctimer_set(&retransmit_timer, (CLOCK_SECOND / 100) *
    retransmission_wait_ms, retransmit_callback, NULL);
+          }
+          else{
+                PRINTF("Sequence is old. Returning.\n");
+                return;
+          }
+
+          /* Update uncomp_hdr_len and rime_hdr_len. */
+          rime_hdr_len += UIP_IPH_LEN;
+          uncomp_hdr_len += UIP_IPH_LEN;
+          break;
     default:
       /* unknown header */
       PRINTFI("sicslowpan input: unknown dispatch: %u\n",
@@ -1845,6 +2014,7 @@ input(void)
       callback->input_callback();
     }

+    PRINTF("Calling TCPIP INPUT\n");
     tcpip_input();
 #if SICSLOWPAN_CONF_FRAG
   }
diff --git a/core/net/sicslowpan.h b/core/net/sicslowpan.h
index 9905fb7..4aa2962 100644
--- a/core/net/sicslowpan.h
+++ b/core/net/sicslowpan.h
@@ -69,6 +69,7 @@
 #define SICSLOWPAN_COMPRESSION_IPV6        0
 #define SICSLOWPAN_COMPRESSION_HC1         1
 #define SICSLOWPAN_COMPRESSION_HC06        2
+#define SICSLOWPAN_COMPRESSION_BC0                 3
 /** @} */
```

```
 /**
@@ -80,6 +81,7 @@
 #define SICSLOWPAN_DISPATCH_IPHC                    0x60 /* 011xxxxx = ...
     */
 #define SICSLOWPAN_DISPATCH_FRAG1                   0xc0 /* 11000xxx */
 #define SICSLOWPAN_DISPATCH_FRAGN                   0xe0 /* 11100xxx */
+#define SICSLOWPAN_DISPATCH_BC0
            0x50 /* 01010000 = 80 */
 /** @} */

 /** \name HC1 encoding
@@ -173,6 +175,7 @@
 #define SICSLOWPAN_HC1_HC_UDP_HDR_LEN               7
 #define SICSLOWPAN_FRAG1_HDR_LEN                    4
 #define SICSLOWPAN_FRAGN_HDR_LEN                    5
+#define SICSLOWPAN_BC0_HDR_LEN                                  2
 /** @} */

 /**
@@ -321,5 +324,20 @@ int sicslowpan_get_last_rssi(void);

 extern const struct network_driver sicslowpan_driver;

+/*---------------------------------------------------------------------------*/
+/* LOWPAN_BC0 structures
    */
+/*---------------------------------------------------------------------------*/
+
+#define LOWPAN_BC0_SEQUENCE_BUF_LEN 20
+
+struct ipv6_sequence{
+       uip_ip6addr_t srcip;
+       uint8_t sequence;
+};
+
+static void
+send_packet(rimeaddr_t *dest);
 #endif /* SICSLOWPAN_H_ */
 /** @} */
+
+
diff --git a/core/net/uip.h b/core/net/uip.h
index c4df7cf..64a9c35 100644
--- a/core/net/uip.h
+++ b/core/net/uip.h
@@ -2095,6 +2095,16 @@ CCIF extern uip_lladdr_t uip_lladdr;
   (((a)->u8[0]) == 0xFF)

 /**
+ * \brief is address a site-local multicast address, see RFC 3513
+ * a is of type uip_ipaddr_t*
+ * */
+#define uip_is_addr_mcast_site_local(a)                   \
```

```
+  (((a)->u8[1] & 0x05) ==  0x05)
+
+#define uip_is_addr_mcast_transient(a)                    \
+  (((a)->u8[1] >> 4 ) ==  0x01)
+
+/**
  * \brief is group-id of multicast address a
  * the all nodes group-id
  */
diff --git a/core/net/uip6.c b/core/net/uip6.c
index aa1ac4a..4c08575 100644
--- a/core/net/uip6.c
+++ b/core/net/uip6.c
@@ -78,14 +78,23 @@

 #include <string.h>

+#ifdef UIP_FALLBACK_INTERFACE
+extern struct uip_fallback_interface UIP_FALLBACK_INTERFACE;
+#endif
+
 #if UIP_CONF_IPV6
 /*---------------------------------------------------------------------------*/
 /* For Debug, logging, statistics                                            */
 /*---------------------------------------------------------------------------*/

-#define DEBUG DEBUG_NONE
+#define DEBUG 0
 #include "net/uip-debug.h"

+#if DEBUG
+#define PRINTF(...) printf(__VA_ARGS__)
+#define PRINT6ADDR(addr)
    PRINTF("[%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x]",
    ((uint8_t *)addr)[0], ((uint8_t *)addr)[1], ((uint8_t *)addr)[2],
    ((uint8_t *)addr)[3], ((uint8_t *)addr)[4], ((uint8_t *)addr)[5],
    ((uint8_t *)addr)[6], ((uint8_t *)addr)[7], ((uint8_t *)addr)[8],
    ((uint8_t *)addr)[9], ((uint8_t *)addr)[10], ((uint8_t *)addr)[11],
    ((uint8_t *)addr)[12], ((uint8_t *)addr)[13], ((uint8_t *)addr)[14],
    ((uint8_t *)addr)[15])
+#endif /* DEBUG */
+
 #if UIP_CONF_IPV6_RPL
 #include "rpl/rpl.h"
 #endif /* UIP_CONF_IPV6_RPL */
@@ -556,9 +564,10 @@ uip_udp_new(const uip_ipaddr_t *ripaddr, uint16_t rport)
   }

   if(conn == 0) {
+
     return 0;
   }
```

```
-
+
   conn->lport = UIP_HTONS(lastport);
   conn->rport = rport;
   if(ripaddr == NULL) {
@@ -1155,11 +1165,11 @@ uip_process(uint8_t flag)
   /* TBD Some Parameter problem messages */
   if(!uip_ds6_is_my_addr(&UIP_IP_BUF->destipaddr) &&
      !uip_ds6_is_my_maddr(&UIP_IP_BUF->destipaddr)) {
-    if(!uip_is_addr_mcast(&UIP_IP_BUF->destipaddr) &&
+    if( (!uip_is_addr_mcast(&UIP_IP_BUF->destipaddr) &&
        !uip_is_addr_link_local(&UIP_IP_BUF->destipaddr) &&
        !uip_is_addr_link_local(&UIP_IP_BUF->srcipaddr) &&
        !uip_is_addr_unspecified(&UIP_IP_BUF->srcipaddr) &&
-       !uip_is_addr_loopback(&UIP_IP_BUF->destipaddr)) {
+       !uip_is_addr_loopback(&UIP_IP_BUF->destipaddr))  ) {


       /* Check MTU */
@@ -1197,6 +1207,15 @@ uip_process(uint8_t flag)
                                ICMP6_DST_UNREACH_NOTNEIGHBOR, 0);
         goto send;
       }
+
+      // check for site local transient multicast address --> forward to
    FALLBACK INTERFACE to transmit
+      // to backhaul link
+#ifdef UIP_FALLBACK_INTERFACE
+      PRINT6ADDR(&UIP_IP_BUF->destipaddr);
+      if(uip_is_addr_mcast(&UIP_IP_BUF->destipaddr) &&
    uip_is_addr_mcast_site_local(&UIP_IP_BUF->destipaddr) &&
    uip_is_addr_mcast_transient(&UIP_IP_BUF->destipaddr)){
+        UIP_FALLBACK_INTERFACE.output();
+      }
+#endif /* !UIP_FALLBACK_INTERFACE */
       PRINTF("Dropping packet, not for me and link local or multicast\n");
       UIP_STAT(++uip_stat.ip.drop);
       goto drop;
```

Glossary

IEEE 802.15.4 IEEE wireless standard for LR-WPANs, aimed at low-rate, low-power appliances [17] (see Section 2.3). 8, 9, 23, 24, 32, 37, 41, 55, 64, 65

IEEE EUI-64 64-bit extended unique identifier for network interfaces or hardware, assigned by the IEEE *Registration Authority*. 9

6LoWPAN IPv6 adaptation layer specification for IEEE 802.15.4 [6] (see Section 2.5). 4, 8–10, 14, 24, 28–32, 35, 37, 39, 41, 43, 44, 78, 79

b6loWPAN 6LoWPAN implementation for TinyOS. 23

Contiki Open source operating system for networked, constrained devices [114] (see Section 5.1.1). x, 4, 14, 17, 26, 28, 30, 37–42, 44, 48, 49, 53, 54, 63, 68, 70, 72, 78, 81, 84

ContikiMAC Radio Duty Cycling (RDC) protocol, implemented in Contiki [12] (see Section 2.2). 17, 43, 53, 72

Cooja Network simulation tool for contiki motes, part of the contiki code base. 4, 37, 44, 47–49, 53

Erbium CoAP Engine for Contiki (see Section 5.1.2). x, 37–39, 41–45, 53–55, 63, 67, 68, 72, 81

IoTSyS IoT integration middleware [101]. 27, 38–40

KNX Industry standard for HBA [115]. 25, 27, 33, 40

Linux Open source operating system, mostly defined through its kernel [116]. 44

LOWPAN_BC0 Broadcast header defined for 6LoWPAN [7], in this thesis also synonymously used for simple multicast engine to forward the datagrams between nodes (see Section 5.1.4). 35, 39, 43, 68, 70, 71, 73, 74, 84

OMNeT++ Open source network simulation framework [113]. 29, 30, 78

powertrace Power usage tracing framework in Contiki [110] (see Section 6.1.1). 49, 51, 52

Protocol Buffers Binary data serialization library [69]. 26, 27

TinyOS Open source operating system for wireless, constrained devices, similar to Contiki [117] (cf. Section 5.1.1). 21, 22, 24, 26

Trickle IPv6 Routing Protocol for Low-Power and Lossy Networks [25] (see Section 2.7). ix, 13, 14, 16, 29, 73

WS-* Set of Web Services specifications, known as *WS star*, closely related to XML, SOAP, WSDL and others. 24

ZigBee Wirless Personal Area Network (WPAN) specification based on IEEE 802.15.4. 8, 22, 23, 25, 26, 29

Acronyms

PAN ID Personal Area Network Identifier. 79

ANSI American National Standards Institute. 26

CCI Channel Check Interval. 15
CDR CompactDataRecord. 24
CIDR Classless Inter-Domain Routing. 8
CoAP Constrained Application Protocol, application layer protocol for constrained devices [6] (see Section 2.10). v, vii, ix, x, 4, 11–13, 16, 21, 24–27, 29, 31–35, 37–41, 43, 44, 47, 54–68, 72, 73, 77–79, 81
CoRE Constrained RESTful Environments, working group in the IETF. v, 11
CPU Central Processing Unit. 26, 49, 51, 52, 55–57, 63–65, 68, 69

DAG Directed Acyclic Graph. 10
DAO Destination Advertisement Object. 11, 28
DIO DODAG Information Object. 10, 11
DODAG Destination Oriented Directed Acyclic Graph. 10, 11, 15, 28, 29, 53, 68, 79
DPWS Devices Profile for Web Services. 21, 25

EXI Efficient XML Interchange, W3C Recommendation for "compressed"/encoded XML [9] (see Section 2.11). ix, 4, 18, 19, 21, 24–27, 38–42, 78

FQDN Fully Qualified Domain Name. 31

HBA Home and Building Automation. 3, 23, 27, 33, 40
HTTP Hypertext Transfer Protocol [32], [118]–[122]. x, 4, 10–13, 24, 25, 37, 53–62, 77

ICMP Internet Control Message Protocol [123], also known as *ICMPv4*. 9
ICMPv6 ICMP version 6 [124], (cf. ICMP). 11, 16, 73
IEEE Institute of Electrical and Electronics Engineers [125]. 8–10, 23, 24, 32, 37, 41, 55, 64, 65
IETF Internet Engineering Task Force. v, vii, 11–13, 24, 27, 34, 38, 77
IoT Internet of Things. v, vii, ix, 1–5, 8, 21, 25, 31–34, 37, 77, 79
IP Internet Protocol, (cf. IPv4 and IPv6). 2–4, 8, 10–13, 21, 24, 25, 28, 29, 33, 54
IPv4 IP version 4 [19]. 8, 37

IPv6 IP version 6 [22] (see Section 2.4). 1, 4, 8, 9, 11, 13–16, 21, 23, 24, 27, 28, 31–34, 38, 39, 41, 43, 44, 51, 53, 70, 79
IQR Interquartile Range. 51, 73
ISO International Organization for Standardization. 7

JSON JavaScript Object Notation. 19, 24, 26

LLN Low power and Lossy Letwork. ix, 10, 13, 14, 16, 17, 28, 35, 37, 70, 79
LR-WPAN Low-Rate Wirless Personal Area Network. 8, 11, 37

M2M Machine to Machine, communication between devices synonymously used for use cases of inter-device communication, e.g. building automation, industrial automation, health monitoring, Smart Grid, etc.. 1, 12, 18
MAODV Multicast Ad Hoc On-Demand Distance Vector. 28
MPL Multicast Protocol for LLNs [30] (see Section 2.8.2). 14–16, 28, 29, 35, 39, 43, 70, 72–75
MQTT Message Queuing Telemetry Transport. 22
MTU Maximum Transmission Unit. 9

NAT Network Address Translation. 8
NDP Neighbor Discovery (ND) Protocol. 44

OASIS Organization for the Advancement of Structured Information Standards [126]. 17, 21
OBIX Open Building Information Xchange, a standard for standardized representation of data, information and objects in HBA systems, (see Section 2.12). v, vii, ix, 4, 17–19, 21, 26, 32, 38, 40, 41, 77
OSI Open Systems Interconnection. ix, 7, 31

QoS Quality of Service. 8, 21

RAM Random-Access Memory. 26, 43
RDC Radio Duty Cycling [11] (see Section 2.2). ix, 16, 17, 37, 50, 53, 55–57, 60, 61, 63–66, 72, 73, 78
REST Representational State Transfer. v, 9–13, 18, 24–26, 32, 37, 53, 54, 77
RFC Request for Comments, a publication form of the IETF that has standardization characteristics. v, vii, 77
ROM Read-Only Memory. 26, 42, 43
RPC Remote Procedure Call. 22
RPL IPv6 Routing Protocol for Low-Power and Lossy Networks [24] (see Section 2.6). ix, 10, 11, 13–16, 28, 29, 35, 37, 39, 41, 51, 53, 66, 73, 74, 78, 79

SensorML Sensor Model Language. 23, 24
SMRF Stateless Multicast RPL Forwading [27] (see Section 2.8.1). 14–16, 28, 29, 35, 39, 70, 72–75, 79
SNOMC Sensor Node Overlay Multicast [75]. 29
SNVT Standard Network Variable Type. 23
SOA Service-Oriented Architecture. 21

SOAP Simple Object Access Protocol. 21, 25, 26
SOS Sensor Observation Service. 23

TCP Transmission Control Protocol [127] (often synonymously used for TCPv4 [128]). 4, 9, 11, 12, 24, 25, 54
TML Transducer Markup Language. 23

UDGM Unit Disk Graph Medium – Distance Loss. 53
UDP User Datagram Protocol [129]. 9, 11, 14, 24, 34
URI Uniform Resource Identifier. 10, 11, 13, 27, 33
URL Uniform Resource Locator. 10, 32, 54
USB Universal Serial Bus, industry standard defining connectors and communication for data and power transfer between various electronic devices. 43

W3C World Wide Web Consortium [130]. 18
WoT Web of Things. ix, 3, 4, 21
WPAN Wirless Personal Area Network. 8
WS SOAP Web Services. 21, 25
WS4D Web Services for Devices. 25
WSAN Wireless Sensor and Actuator Network. v, ix, 8, 21–25, 27–29, 32, 33, 35, 41, 43, 44, 50, 53, 54, 66, 68, 71, 77–79
WSDL Web Service Description Language. 21

XML Extensible Markup Language, W3C Recommendation [131]. ix, 4, 18, 19, 21, 23–27, 38–42, 77, 78
XMPP eXtensible Messaging and Presence Protocol. 21

Bibliography

[1] O. Hersent, D. Boswarthick, and O. Elloumi, *The Internet of Things*. Chichester, UK: John Wiley & Sons, Ltd, Dec. 2011 (cit. on p. 1).

[2] C. Thompson, "Smart Devices and Soft Controllers", *IEEE Internet Comput.*, vol. 9, no. 1, pp. 82–85, Jan. 2005 (cit. on p. 1).

[3] D. Guinard, "A Web of Things Application Architecture - Integrating the Real-World into the Web", PhD thesis, ETH Zurich, 2011 (cit. on p. 4).

[4] D. Guinard and V. Trifa, "Towards the web of things: Web mashups for embedded devices", in *Proc. 18th Int. World Wide Web Conf. (WWW 2009), Mashups, Enterp. Mashups Light. Compos. Web (MEM 2009 Work.*, 2009 (cit. on pp. 4, 24).

[5] Z. Shelby, "Embedded web services", *IEEE Wirel. Commun.*, vol. 17, no. 6, pp. 52–57, Dec. 2010 (cit. on pp. 4, 26).

[6] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)", *IETF RFC 7252*, Jun. 2014 (cit. on pp. 5, 16, 18, 38, 55, 72, 93, 95).

[7] G. Montenegro, N. Kushalnagar, J. W. Hui, and D. E. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", *IETF RFC 4944*, Sep. 2007 (cit. on pp. 5, 9, 10, 39, 44, 93).

[8] T. Considine and P. Ehrlich, "oBIX 1.0", *OASIS CS*, Dec. 2006 (cit. on pp. 5, 19, 20, 32).

[9] *Efficient XML Interchange (EXI) Format 1.0 (Second Edition)*, Feb. 2014. [Online]. Available: http://www.w3.org/TR/2014/REC-exi-20140211/ (visited on 05/18/2014) (cit. on pp. 5, 18, 40, 95).

[10] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-Level Sensor Network Simulation with COOJA", in *Proc. 31st IEEE Conf. Local Comput. Networks*, IEEE, Nov. 2006, pp. 641–648 (cit. on pp. 6, 48).

[11] A. Dunkels, L. Mottola, and N. Tsiftes, "The announcement layer: Beacon coordination for the sensornet stack", in *Proc. 8th Eur. Conf. Wirel. Sens. Networks*, P. Marrón and K. Whitehouse, Eds., ser. LNCS, vol. 6567, Springer, Feb. 2011, pp. 211–226 (cit. on pp. 8, 96).

[12] A. Dunkels, "The ContikiMAC Radio Duty Cycling Protocol", Tech. Rep., Dec. 2011 (cit. on pp. 8, 53, 57, 93).

[13] J. Polastre, J. Hill, and D. Culler, “Versatile low power media access for wireless sensor networks”, in *Proc. 2nd Int. Conf. Embed. networked Sens. Syst. (SenSys ’04)*, New York, New York, USA: ACM Press, Nov. 2004, pp. 95–107 (cit. on p. 8).

[14] M. Buettner, G. V. Yee, E. Anderson, and R. Han, “X-MAC: A Short Preamble MAC Protocol for Duty-Cycled Wireless Sensor Networks”, in *Proc. 4th Int. Conf. Embed. networked Sens. Syst. - SenSys ’06*, New York, New York, USA: ACM Press, Oct. 2006, pp. 307–320 (cit. on p. 8).

[15] D. Moss and P. Levis, “BoX-MACs: Exploiting physical and link layer boundaries in low-power networking”, Tech. Rep., 2008 (cit. on p. 8).

[16] A. El-Hoiydi and J.-D. Decotignie, “WiseMAC: an ultra low power MAC protocol for the downlink of infrastructure wireless sensor networks”, in *Proc. 9th Int. Symp. Comput. Commun. (ISCC 2004)*, vol. 1, IEEE, Jul. 2004, pp. 244–251 (cit. on p. 8).

[17] *Low-Rate Wireless Personal Area Networks (LR-WPANs)*, 2011. [Online]. Available: http://standards.ieee.org/findstds/standard/802.15.4-2011.html (visited on 05/30/2014) (cit. on pp. 9, 93).

[18] J. Gutierrez, M. Naeve, E. Callaway, M. Bourgeois, V. Mitter, and B. Heile, “IEEE 802.15.4: a developing standard for low-power low-cost wireless personal area networks”, *IEEE Netw.*, vol. 15, no. 5, pp. 12–19, Sep. 2001 (cit. on p. 9).

[19] “Internet Protocol”, *IETF RFC 791*, Sep. 1981 (cit. on pp. 9, 95).

[20] S. E. Deering and R. M. Hinden, “Internet Protocol, Version 6 (IPv6) Specification”, *IETF RFC 1883*, Dec. 1995 (cit. on p. 9).

[21] N. Leavitt, “IPv6: Any Closer to Adoption?”, *Computer (Long. Beach. Calif).*, vol. 44, no. 9, pp. 14–16, Sep. 2011 (cit. on p. 9).

[22] S. E. Deering and R. M. Hinden, “Internet Protocol, Version 6 (IPv6) Specification”, *IETF RFC 2460*, Dec. 1998 (cit. on pp. 10, 96).

[23] M. Crawford, “Transmission of IPv6 Packets over Ethernet Networks”, *IETF RFC 2464*, Dec. 1998 (cit. on p. 10).

[24] T. Winter, A. Brandt, J. W. Hui, R. Kelsey, P. Levis, K. S. J. Pister, J. Vasseur, and R. K. Alexander, “RPL: IPv6 routing protocol for low-power and lossy networks”, *IETF RFC 6550*, Mar. 2012 (cit. on pp. 10, 28, 96).

[25] P. Levis, T. H. Clausen, J. W. Hui, O. Gnawali, and J. Ko, “The Trickle Algorithm”, *IETF RFC 6206*, Mar. 2011 (cit. on pp. 12, 94).

[26] P. Levis, N. Patel, D. Culler, and S. Shenker, “Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks”, in *Proc. 1st Symp. Networked Syst. Des. Implement. (NSDI ’04)*, USENIX Association, 2004, pp. 15–28 (cit. on p. 12).

[27] G. Oikonomou and I. Phillips, “Stateless multicast forwarding with RPL in 6LowPAN sensor networks”, in *Proc. Int. Conf. Pervasive Comput. Commun. Work.*, IEEE, Mar. 2012, pp. 272–277 (cit. on pp. 13–15, 28, 72, 74, 96).

[28] T. Clausen and U. Herberg, “Comparative study of RPL-enabled optimized broadcast in Wireless Sensor Networks”, in *Proc. 6th Int. Conf. Intell. Sensors, Sens. Networks Inf. Process.*, Laboratoire d’Informatique (LIX) - Ecole Polytechnique, France, IEEE, Dec. 2010, pp. 7–12 (cit. on pp. 13, 28).

[29] G. Oikonomou, I. Phillips, and T. Tryfonas, “IPv6 Multicast Forwarding in RPL-Based Wireless Sensor Networks”, *Wirel. Pers. Commun.*, vol. 73, no. 3, pp. 1089–1116, Jun. 2013 (cit. on pp. 14, 28).

[30] J. Hui and R. Kelsey, “Multicast Protocol for Low power and Lossy Networks (MPL)”, *IETF Internet Draft. Work Prog.*, Apr. 2014 (cit. on pp. 14, 96).

[31] R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures”, PhD thesis, University of California, Irvine, 2000, pp. 76–106 (cit. on p. 15).

[32] R. T. Fielding and J. F. Reschke, “Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content”, *IETF RFC 7231*, 2014 (cit. on pp. 15, 95).

[33] M. Nottingham and E. Hammer-Lahav, “Defining Well-Known Uniform Resource Identifiers (URIs)”, *IETF RFC 5785*, Apr. 2010 (cit. on p. 16).

[34] C. Bormann and Z. Shelby, “Blockwise transfers in CoAP”, *IETF Internet Draft. Work Prog.*, Jul. 2014 (cit. on pp. 17, 38).

[35] K. Hartke, “Observing Resources in CoAP”, *IETF Internet Draft. Work Prog.*, Jan. 2014 (cit. on pp. 17, 38).

[36] I. Ishaq, J. Hoebeke, F. Van Den Abeele, I. Moerman, and P. Demeester, “Group Communication in Constrained Environments Using CoAP-based Entities”, in *Proc. 2013 IEEE Int. Conf. Distrib. Comput. Sens. Syst.*, IEEE, May 2013, pp. 345–350 (cit. on pp. 18, 27).

[37] M. Jung and W. Kastner, “Efficient group communication based on Web services for reliable control in wireless automation”, in *Proc. 39th Annu. Conf. IEEE Ind. Electron. Soc. (IECON 2013)*, IEEE, Nov. 2013, pp. 5716–5722 (cit. on pp. 18, 27, 34).

[38] A. Rahman and E. Dijk, “Group Communication for CoAP”, *IETF Internet Draft. Work Prog.*, Jul. 2014 (cit. on pp. 18, 33).

[39] M. Jung, “Encodings for OBIX: Common Encodings Version 1.0”, *OASIS CS Draft 02 / Public Rev. Draft 02*, vol. 2013, (cit. on p. 19).

[40] P. Nie and J. K. Nurminen, “Integrate WSN to the Web of Things by Using XMPP”, in *Sens. Syst. Softw.* Ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, F. Martins, L. Lopes, and H. Paulino, Eds., vol. 102, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 105–120 (cit. on p. 21).

[41] “OASIS Devices Profile for Web Services (DPWS)”, *OASIS Stand.*, Jul. 2009 (cit. on p. 21).

[42] G. Moritz, F. Golatowski, C. Lerche, and D. Timmermann, “Beyond 6LoWPAN: Web Services in Wireless Sensor Networks”, *IEEE Trans. Ind. Informatics*, vol. 9, no. 4, pp. 1795–1805, Nov. 2013 (cit. on pp. 21, 25, 26).

[43] G. F. Anastasi, E. Bini, A. Romano, and G. Lipari, “A service-oriented architecture for QoS configuration and management of Wireless Sensor Networks”, in *Proc. 15th Conf. Emerg. Technol. Fact. Autom. (ETFA 2010)*, IEEE, Sep. 2010, pp. 1–8 (cit. on pp. 21, 25).

[44] E. Souto, G. Guimarães, G. Vasconcelos, M. Vieira, N. Rosa, and C. Ferraz, “A message-oriented middleware for sensor networks”, in *Proc. 2nd Work. Middlew. pervasive ad-hoc Comput.*, Toronto, Ontario, Canada: ACM Press, Oct. 2004, pp. 127–134 (cit. on p. 22).

[45] E. Souto, G. Guimarães, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz, and J. Kelner, “Mires: a publish/subscribe middleware for sensor networks”, *Pers. Ubiquitous Comput.*, vol. 10, no. 1, pp. 37–44, Oct. 2005 (cit. on p. 22).

[46] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, “TinyDB: an acquisitional query processing system for sensor networks”, *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 122–173, Mar. 2005 (cit. on p. 22).

[47] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, “MQTT-S — A publish/subscribe protocol for Wireless Sensor Networks”, in *Proc. 3rd Int. Conf. Commun. Syst. Softw. Middlew. Work. (COMSWARE ’08)*, IEEE, Jan. 2008, pp. 791–798 (cit. on p. 22).

[48] M. Navarro, M. Antonucci, L. Sarakis, and T. Zahariadis, “VITRO Architecture: Bringing Virtualization to WSN World”, in *Proc. 8th Int. Conf. Mob. Ad-Hoc Sens. Syst.*, IEEE, Oct. 2011, pp. 831–836 (cit. on p. 22).

[49] P. Kostelnik, M. Sarnovsk, and K. Furdik, “The semantic middleware for networked embedded systems applied in the Internet of Things and Services domain”, *Scalable Comput. Pract. Exp.*, vol. 12, no. 3, pp. 307–315, 2011 (cit. on p. 22).

[50] D. Hughes, K. Thoelen, W. Horré, N. Matthys, J. D. Cid, S. Michiels, C. Huygens, and W. Joosen, “LooCI: a loosely-coupled component infrastructure for networked embedded systems”, in *Proc. 7th Int. Conf. Adv. Mob. Comput. Multimed. - MoMM ’09*, New York, New York, USA: ACM Press, 2009, p. 195 (cit. on p. 22).

[51] K. Menzel, D. Pesch, B. O’Flynn, M. Keane, and C. O’Mathuna, “Towards a wireless sensor platform for energy efficient building operation”, *Tsinghua Sci. Technol.*, vol. 13, no. S1, pp. 381–386, Oct. 2008 (cit. on p. 23).

[52] M. Jung, J. Weidinger, W. Kastner, and A. Olivieri, “Heterogeneous device interaction using an IPv6 enabled service-oriented architecture for building automation systems”, in *Proc. 28th Annu. ACM Symp. Appl. Comput. (SAC ’13)*, New York, New York, USA: ACM Press, Mar. 2013, pp. 1939–1941 (cit. on p. 23).

[53] M. Botts, G. Percivall, C. Reed, J. Davidson, I. Architecture, O. G. Consortium, and I. Matters, “OGC® Sensor Web Enablement: Overview and High Level Architecture”, in *GeoSensor Networks*, ser. Lecture Notes in Computer Science, vol. 4540, Springer Berlin Heidelberg, 2008, pp. 175–190 (cit. on pp. 23, 25).

[54] A. Triantafyllidis, V. G. Koutkias, I. Chouvarda, and N. M. Maglaveras, “An open and reconfigurable Wireless Sensor Network for pervasive health monitoring”, in *Proc. 2nd Int. Conf. Pervasive Comput. Technol. Healthc.*, vol. 47, IEEE, Jan. 2008, pp. 112–115 (cit. on p. 23).

[55] K. Aberer, M. Hauswirth, and A. Salehi, “A Middleware for Fast and Flexible Sensor Network Deployment”, in *Proc. 32nd Int. Conf. Very large data bases (VLDB ’06)*, VLDB Endowment, Sep. 2006, pp. 1199–1202 (cit. on pp. 23, 25).

[56] R. Špinar, P. Muthukumaran, R. de Paz, D. Pesch, W. Song, S. A. Chaudhry, C. J. Sreenan, E. Jafer, B. O’Flynn, J. O’Donnell, A. Costa, and M. M. Keane, “Efficient building management with IP-based wireless sensor network”, in *Proc. 6th Eur. Conf. Wirel. Sens. Networks*, Demo abstract, Cork, Ireland, Feb. 2009, pp. 8–10 (cit. on p. 23).

[57] C. Conway and M. Barnes, “Wireless sensor network data description and encoding in heterogeneous building systems”, in *Proc. Int. Conf. Distrib. Comput. Sens. Syst. Work.*, IEEE, Jun. 2011, pp. 1–4 (cit. on pp. 24, 27).

[58] A. Azzara, S. Bocchino, P. Pagano, G. Pellerano, and M. Petracca, “Middleware solutions in WSN: The IoT oriented approach in the ICSI project”, in *Proc. 21st Int. Conf. Software, Telecommun. Comput. Networks - (SoftCOM 2013)*, IEEE, Sep. 2013, pp. 1–6 (cit. on pp. 24, 25).

[59] S. Dawson-Haggerty, X. Jiang, G. Tolle, J. Ortiz, and D. Culler, “sMAP – a Simple Measurement and Actuation Profile for Physical Information”, in *Proc. 8th ACM Conf. Embed. Networked Sens. Syst. - SenSys ’10*, New York, New York, USA: ACM Press, Nov. 2010, p. 197 (cit. on p. 24).

[60] A. Ludovici, P. Moreno, and A. Calveras, “TinyCoAP: A Novel Constrained Application Protocol (CoAP) Implementation for Embedding RESTful Web Services in Wireless Sensor Networks Based on TinyOS”, *J. Sens. Actuator Networks*, vol. 2, no. 2, pp. 288–315, May 2013 (cit. on pp. 24, 38).

[61] D. Guinard, V. Trifa, and E. Wilde, “A resource oriented architecture for the Web of Things”, in *Proc. Internet of Things (IOT)*, IEEE, Nov. 2010, pp. 1–8 (cit. on p. 24).

[62] D. Guinard, M. Fischer, and V. Trifa, “Sharing using social networks in a composable Web of Things”, in *Proc. 8th IEEE Int. Conf. Pervasive Comput. Commun. Work. (PERCOM Work.*, IEEE, Mar. 2010, pp. 702–707 (cit. on p. 24).

[63] J. Helander, “Deeply embedded XML communication”, in *Proc. 5th ACM Int. Conf. Embed. Softw. - EMSOFT ’05*, New York, New York, USA: ACM Press, 2005, p. 62 (cit. on pp. 25, 42).

[64] G. Moritz, F. Golatowski, and D. Timmermann, “A Lightweight SOAP over CoAP Transport Binding for Resource Constraint Networks”, in *Proc. 8th Int. Conf. Mob. Ad-Hoc Sens. Syst.*, IEEE, Oct. 2011, pp. 861–866 (cit. on p. 25).

[65] A. P. Castellani, M. Gheda, N. Bui, M. Rossi, and M. Zorzi, “Web Services for the Internet of Things through CoAP and EXI”, in *Proc. IEEE Int. Conf. Commun. Work.*, IEEE, Jun. 2011 (cit. on p. 26).

[66] D. Caputo, L. Mainetti, L. Patrono, and A. Vilei, “Implementation of the EXI Schema on Wireless Sensor Nodes Using Contiki”, in *Proc. 6th Int. Conf. Innov. Mob. Internet Serv. Ubiquitous Comput.*, IEEE, Jul. 2012, pp. 770–774 (cit. on pp. 26, 40).

[67] *EXIP - Embeddable EXI Processor*. [Online]. Available: http://exip.sourceforge.net/ (visited on 05/17/2014) (cit. on p. 26).

[68] S. Käbisch, D. Peintner, J. Heuer, and H. Kosch, “Efficient and Flexible XML-Based Data-Exchange in Microcontroller-Based Sensor Actor Networks”, in *Proc. 24th Int. Conf. Adv. Inf. Netw. Appl. Work.*, IEEE, 2010, pp. 508–513 (cit. on pp. 26, 42).

[69] G. Inc., *Protocol Buffers — Google Developers*, Jun. 2014. [Online]. Available: https://developers.google.com/protocol-buffers/ (visited on 08/27/2014) (cit. on pp. 26, 93).

[70] N. Gligoric, I. Dejanovic, and S. Krco, “Performance evaluation of compact binary XML representation for constrained devices”, in *2011 Int. Conf. Distrib. Comput. Sens. Syst. Work.*, IEEE, Jun. 2011 (cit. on p. 27).

[71] M. Jung, P. Raich, and W. Kastner, “The relevance and impact of IPv6 multicasting for Wireless Sensor and Actuator Networks based on 6LoWPAN and Constrained RESTful Environments”, in *Proc. 4th Int. Conf. Internet Things (IoT 2014)*, Cambridge, MA, USA, Oct. 2014 (cit. on p. 28).

[72] J. S. Silva, T. Camilo, P. Pinto, R. Ruivo, A. Rodrigues, F. Gaudêncio, and F. Boavida, “Multicast and IP Multicast support in Wireless Sensor Networks”, *JNW*, vol. 3, no. 3, pp. 19–26, Mar. 2008 (cit. on p. 28).

[73] K. Tharatipayakul, S. Gordon, and K. Kaemarungsi, “iACK: Implicit acknowledgements to improve multicast reliability in wireless sensor networks”, in *Proc. 11th Int. Conf. Electr. Eng. Comput. Telecommun. Inf. Technol.*, IEEE, May 2014, pp. 1–6 (cit. on p. 29).

[74] G. Wagenknecht, M. Anwander, and T. Braun, “SNOMC: An overlay multicast protocol for Wireless Sensor Networks”, in *Proc. 9th Annu. Conf. Wirel. On-Demand Netw. Syst. Serv. (WONS 2009)*, IEEE, Jan. 2012, pp. 75–78 (cit. on p. 29).

[75] G. Wagenknecht and T. I. Braun, *Internet of Things, Smart Spaces, and Next Generation Networks and Systems*, S. Balandin, S. Andreev, and Y. Koucheryavy, Eds., ser. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2014, vol. 8638, pp. 280–293 (cit. on pp. 29, 96).

[76] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, “Directed diffusion for wireless sensor networking”, *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 2–16, Feb. 2003 (cit. on p. 29).

[77] O. Liang, Y. Sekercioglu, and N. Mani, “A survey of multipoint relay based broadcast schemes in wireless ad hoc networks”, *IEEE Commun. Surv. Tutorials*, vol. 8, no. 4, pp. 30–46, Jan. 2006 (cit. on p. 29).

[78] A. Marchiori and Q. Han, “PIM-WSN: Efficient multicast for IPv6 wireless sensor networks”, in *Proc. Int. Symp. a World Wireless, Mob. Multimed. Networks*, IEEE, Jun. 2011, pp. 1–6 (cit. on p. 29).

[79] Q. Cao, T. He, and T. Abdelzaher, “uCast: Unified Connectionless Multicast for Energy Efficient Content Distribution in Sensor Networks”, *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 2, pp. 240–250, Feb. 2007 (cit. on p. 29).

[80] C.-y. Wan, A. T. Campbell, and L. Krishnamurthy, “PSFQ”, in *Proc. 1st ACM Int. Work. Wirel. Sens. networks Appl. - WSNA ’02*, New York, New York, USA: ACM Press, 2002, p. 1 (cit. on p. 29).

[81] P. Marron, A. Lachenmann, D. Minder, J. Hahner, R. Sauter, and K. Rothermel, “TinyCubus: a flexible and adaptive framework sensor networks”, in *Proc. 2nd Eur. Work. Wirel. Sens. Networks*, IEEE, Feb. 2005, pp. 278–289 (cit. on p. 29).

[82] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele, “Low-power wireless bus”, in *Proc. 10th ACM Conf. Embed. Netw. Sens. Syst. - SenSys ’12*, New York, New York, USA: ACM Press, 2012, p. 1 (cit. on p. 29).

[83] M. Doddavenkatappa, M. C. Chan, and B. Leong, “Splash : Fast Data Dissemination with Constructive Interference in Wireless Sensor Networks”, in *Proc. 10th USENIX Symp. Networked Syst. Des. Implement. (NSDI 13)*, Apr. 2013, pp. 269–282 (cit. on p. 29).

[84] S. Banerjee, A. Misra, and A. Agrawala, “Energy-efficient broadcast and multicast trees for reliable wireless communication”, in *Proc. Wirel. Commun. Networking, WCNC 2003*, vol. 1, IEEE, Mar. 2003, pp. 660–667 (cit. on p. 29).

[85] X. Wang, Y. Tan, Z. Zhou, and L. Zhou, “Reliable and energy-efficient multicast based on network coding for wireless sensor networks”, in *Proc. 6th Int. ICST Conf. Commun. Netw. China*, vol. 1, IEEE, Aug. 2011, pp. 1142–1145 (cit. on p. 29).

[86] O. Gaddour, A. Koubaa, O. Cheikhrouhou, and M. Abid, “Z-Cast: A Multicast Routing Mechanism in ZigBee Cluster-Tree Wireless Sensor Networks”, in *Proc. 30th Int. Conf. Distrib. Comput. Syst. Work.*, IEEE, Jun. 2010, pp. 171–179 (cit. on p. 29).

[87] H. Boujelben, O. Gaddour, and M. Abid, “Enhancement and performance evaluation of a multicast routing mechanism in ZigBee cluster-tree Wireless Sensor networks”, in *Proc.10th Int. Multi-Conferences Syst. Signals Devices 2013*, IEEE, Mar. 2013 (cit. on p. 29).

[88] D. Bunyai, L. Krammer, and W. Kastner, “Limiting constraints for ZigBee networks”, in *Proc. 38th Annu. Conf. IEEE Ind. Electron. Soc. (IECON 12)*, IEEE, Oct. 2012, pp. 4840–4846 (cit. on p. 29).

[89] M. Kirsche and J. Hartwig, “A 6LoWPAN Model for OMNeT++”, in *Proc. 6th Int. ICST Conf. Simul. Tools Tech.*, Mar. 2013, pp. 3–6 (cit. on p. 30).

[90] P. Liggesmeyer, “Einleitung und Überblick”, in *Softw. Eng. eingebetteter Syst. Grundlagen – Method. – Anwendungen*, P. Liggesmeyer and H. D. Rombach, Eds., Elsevier, Spektrum Akademischer Verlag, 2005, pp. 1–12 (cit. on p. 37).

[91] A. Dunkels, B. Grönvall, and T. Voigt, “Contiki - a lightweight and flexible operating system for tiny networked sensors”, in *29th Annu. IEEE Int. Conf. Local Comput. Networks*, IEEE (Comput. Soc.), Nov. 2004, pp. 455–462 (cit. on p. 37).

[92] D. Yazar and A. Dunkels, “Efficient application integration in IP-based sensor networks”, *Proc. 1st ACM Work. Embed. Sens. Syst. EnergyEfficiency Build. BuildSys 09*, pp. 43–48, Nov. 2009 (cit. on p. 37).

[93] D. Yazar, “RESTful Wireless Sensor Networks”, Master’s thesis, Dept. Information Technology, Uppsala Universitet, Oct. 2009 (cit. on p. 37).

[94] M. Kovatsch, S. Duquennoy, and A. Dunkels, “A Low-Power CoAP for Contiki”, in *Proc. 8th Int. Conf. Mob. Ad-Hoc Sens. Syst.*, IEEE, Oct. 2011, pp. 855–860 (cit. on p. 37).

[95] Z. Shelby, B. Frank, and D. Sturek, “Constrained Application Protocol (CoAP)”, *IETF Internet Draft. Work Prog.*, Oct. 2009 (cit. on p. 38).

[96] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, “Constrained Application Protocol (CoAP)”, *IETF Internet Draft. Work Prog.*, Jul. 2011 (cit. on p. 38).

[97] *Californium (Cf) CoAP framework - Java CoAP Implementation*. [Online]. Available: http://people.inf.ethz.ch/mkovatsc/californium.php (visited on 07/18/2014) (cit. on p. 38).

[98] *CoAPy: Constrained Application Protocol in Python — CoAPy v0.0.2 documentation*. [Online]. Available: http://coapy.sourceforge.net/ (visited on 07/18/2014) (cit. on p. 38).

[99] *jcoap - jCoAP is a Java Library implementing the Constrained Application Protocol (CoAP) - Google Project Hosting*. [Online]. Available: https://code.google.com/p/jcoap/ (visited on 07/08/2014) (cit. on p. 38).

[100] K. Kuladinithi, O. Bergmann, T. Pötsch, M. Becker, and C. Görg, “Implementation of coap and its application in transport logistics”, in *Proc. 10th Int. Conf. Inf. Process. Sens. Networks*, Chicago, IL, Apr. 2011 (cit. on p. 38).

[101] M. Jung, J. Chelakal, J. Schober, W. Kastner, L. Zhou, and G. K. Nam, “IoTSyS: an integration middleware for the Internet of Things”, in *Proc. 4th Int. Conf. Internet Things (IoT 2014)*, Demo abstract, Cambridge, MA, USA, Oct. 2014 (cit. on pp. 38, 93).

[102] *IoTSyS - Internet of Things integration middleware*. [Online]. Available: https://code.google.com/p/iotsys/ (visited on 07/05/2014) (cit. on p. 38).

[103] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu, “The broadcast storm problem in a mobile ad hoc network”, in *Proc. 5th Annu. Int. Conf. Mob. Comput. Netw. - MobiCom ’99*, vol. 8, New York, New York, USA: ACM Press, 1999, pp. 151–162 (cit. on p. 39).

[104] M. Jung, J. Weidinger, and W. Kastner, “A Seamless Integration of KNX into Constrained RESTful Environments”, in *Proc. KNX Sci.*, Las Palmas, Spain, Nov. 2012 (cit. on p. 41).

[105] *Products | zolertia*. [Online]. Available: http://zolertia.com/products (visited on 05/18/2014) (cit. on p. 42).

[106] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, “Constrained Application Protocol (CoAP)”, *IETF Internet Draft. Work Prog.*, Dec. 2012 (cit. on p. 42).

[107] *Linux IPv6 Router Advertisement Daemon (radvd)*, Sep. 2014. [Online]. Available: http://www.litech.org/radvd/ (visited on 09/06/2014) (cit. on p. 44).

[108] S. Thomson, T. Narten, and T. Jinmei, "IPv6 Stateless Address Autoconfiguration Status", *IETF RFC 4862*, Sep. 2007 (cit. on p. 44).

[109] *GitHub: Cetic – cooja-radiologger-headless*, Jan. 2014. [Online]. Available: https://github.com/cetic/cooja-radiologger-headless/tree/b970037/java/be/cetic/cooja/plugins (visited on 07/26/2014) (cit. on p. 49).

[110] A. Dunkels, J. Eriksson, N. Finne, and N. Tsiftes, "Powertrace: Network-level Power Profiling for Low-power Wireless Networks", Swedish Institute of Computer Science, Tech. Rep., Mar. 2011 (cit. on pp. 49, 93).

[111] *Curl and libcurl*. [Online]. Available: http://curl.haxx.se/ (visited on 09/29/2014) (cit. on pp. 53, 54).

[112] H. Ali, "A Performance Evaluation of RPL in Contiki A Cooja Simulation based study", Master's thesis, School of Computing, Blekinge Institute of Technology (BTH), 2012 (cit. on p. 66).

[113] *OMNeT++ Network Simulation Framework*. [Online]. Available: http://www.omnetpp.org/ (visited on 09/05/2014) (cit. on pp. 78, 93).

[114] *Contiki: The Open Source Operating System for the Internet of Things*. [Online]. Available: http://www.contiki-os.org/ (visited on 07/07/2014) (cit. on p. 93).

[115] *KNX Association*. [Online]. Available: http://www.knx.org (visited on 09/27/2014) (cit. on p. 93).

[116] *The Linux Kernel Archives*. [Online]. Available: https://www.kernel.org/ (visited on 09/05/2014) (cit. on p. 93).

[117] *TinyOS Home Page*. [Online]. Available: http://www.tinyos.net/ (visited on 07/07/2014) (cit. on p. 93).

[118] R. T. Fielding and J. F. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", Jun. 2014 (cit. on p. 95).

[119] ——, "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", *IETF RFC 7232*, Jun. 2014 (cit. on p. 95).

[120] R. T. Fielding, Y. Lafon, and J. F. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", *IETF RFC 7233*, Jun. 2014 (cit. on p. 95).

[121] R. T. Fielding, M. Nottingham, and J. F. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Caching", *IETF RFC 7234*, Jun. 2014 (cit. on p. 95).

[122] R. T. Fielding and J. F. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Authentication", *IETF RFC 7235*, Jun. 2014 (cit. on p. 95).

[123] J. Postel, "Internet Control Message Protocol", *IETF RFC 792*, Sep. 1981 (cit. on p. 95).

[124] A. Conta, S. Deering, and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", *IETF RFC 4443*, Mar. 2006 (cit. on p. 95).

[125] *IEEE - Institute of Electrical and Electronics Engineers*. [Online]. Available: https://www.ieee.org/index.html (visited on 09/10/2014) (cit. on p. 95).

[126] *OASIS | Advancing open standards for the information society*. [Online]. Available: https://www.oasis-open.org/ (visited on 06/07/2014) (cit. on p. 96).

[127] V. Cerf, Y. Dalal, and C. Sunshine, "Specification Of Internet Transmission Control Program", *IETF RFC 675*, pp. 1–70, Dec. 1974 (cit. on p. 97).

[128] "Transmission Control Protocol", *IETF RFC 793*, Sep. 1981 (cit. on p. 97).

[129] J. Postel, "User Datagram Protocol", *IETF RFC 768*, Aug. 1980 (cit. on p. 97).

[130] *World Wide Web Consortium (W3C)*. [Online]. Available: http://www.w3.org/ (visited on 06/09/2014) (cit. on p. 97).

[131] *Extensible Markup Language (XML)*. [Online]. Available: http://www.w3.org/XML/ (visited on 09/30/2014) (cit. on p. 97).