Pekka Karhula

Internet of Things: a gateway centric solution for providing IoT connectivity

Master's Thesis
in Information Technology
May 26, 2016

University of Jyväskylä

Department of Mathematical Information Technology

Kokkola University Consortium Chydenius

Author: Pekka Karhula
Contact information: pekka.karhula@vtt.fi
Phonenumber: +358 (0)40-830 9162
Supervisor: Ismo Hakala
Title: Internet of Things: a gateway centric solution for providing IoT connectivity
Työn nimi: Esineiden Internet: gateway-pohjainen ratkaisu IoT–yhdistettävyyden toteuttamiseksi
Project: Master's Thesis in Information Technology
Page count: 74
Abstract: The Internet of Things (IoT) is revolutionising the traditional Internet by extending it with smart everyday objects. Wearables, smart grids and home automation systems are just a few examples of the IoT. A noteworthy point is that the amount of devices connected to the Internet will rapidly grow with the IoT. The IoT typically involves devices that are constrained in terms of energy, memory and processing resources. Therefore, they also limit applying the existing Internet protocols to the IoT. New protocols have been designed for the IoT, all the way from the physical layer, to the application layer. This thesis presents an implementation for an IoT gateway, which enables connectivity to the Internet for several different end devices using protocols designed for the IoT. This thesis reviews existing IoT architectures, protocol stacks, IoT gateway functionalities and management, and presents three case examples of gateway usage scenarios.
Suomenkielinen tiivistelmä: Esineiden internet (engl. Internet of Things, IoT) on mullistamassa perinteistä Internetiä laajentamalla sitä älykkäillä, jokapäiväisillä esineillä. Puettava teknologia, älykäs sähköverkko ja kotiautomaatio ovat vain muutama esimerkki esineiden internetistä. Huomionarvoista on, että Internetiin liitettävien laitteiden määrä kasvaa ripeästi esineiden internetin myötä. Esineiden internetin sovelluksissa käytössä on tyypillisesti akkukestoltaan sekä muisti- ja laskentakapasiteetiltaan rajoitettuja laitteita. Siten ne myös rajoittavat nykyisten Internet-protokollien soveltamista esineiden internetiin. Uusia protokollia on kehitetty esineiden internetiä varten aina fyysiseltä tasolta sovelluskerrokseen. Tässä työssä esitetään toteutus IoT-tukiasemalle (engl. IoT gateway), joka mahdollistaa useiden erilaisten laitteiden yhdistämisen Internetiin, käyttämällä esineiden internetiä varten suunniteltuja protokollia. Työssä käydään läpi olemassa olevia IoT-arkkitehtuureja, protokollapinoja, IoT-tukiaseman ominaisuuksia ja hallintaa sekä esitetään kolme case-esimerkkiä, joissa IoT-tukiasemaa on käytetty.

Keywords: Internet of Things, IoT gateway, CoAP, MQTT, Distributed Decision Engine
Avainsanat: Esineiden Internet, IoT tukiasema, CoAP, MQTT, Distributed Decision Engine

Preface

I would like to thank VTT for giving me the opportunity to work on this topic and write a thesis about it. It has been a truly interesting and educational experience. Hopefully, it is able to provide valuable information for interested readers, as well.

I would also like to thank my supervisors, Professor Ismo Hakala from the University of Jyväskylä and Dr Jukka Mäkelä from VTT for their valuable feedback. Also, many thanks to Mirjami Jutila for reading the thesis and giving me important advice.

Glossary

3GPP	3rd Generation Partnership Project
6LoWPAN	IPv6 over Low-power Wireless Personal Area Networks
BLE	Bluetooth Low Energy
CCA	Clear Channel Assessment
CoAP	Constrained Application Protocol
CPE	Customer-Premises Equipment
DDE	Distributed Decision Engine
DTLS	Datagram Transport Layer Security
ED	Energy Detection
ETSI	European Telecommunications Standards Institute
FFD	Full Functional Device
GPRS	General Packet Radio Service
HCI	Human-Computer Interaction
HTTP	HyperText Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IoE	Internet of Everything (also Internet of Energy)
IoT	Internet of Things
IDE	Integrated Development Environment
IP	Internet Protocol
JSON	JavaScript Object Notation
LLN	Low-power and Lossy Network
LPWA	Low-Power Wide-Area
LQI	Link Quality Indicator
LTE	Long-Term Evolution
LWM2M	Lightweight Machine to Machine

M2M	Machine to Machine
MAC	Medium Access Control
MQTT	MQ Telemetry Transport
MTC	Machine-Type Communication
NFV	Network Function Virtualisation
OMA	Open Mobile Alliance
OMA-DM	Open Mobile Alliance Device Management
OSI	Open Systems Interconnection
OUI	Organisational Unique Identifier
PAN	Personal Area Network
PPDU	PHY Protocol Data Unit
REST	Representational State Transfer
RFC	Request For Comments
RFD	Reduced Function Device
RFID	Radio-Frequency Identification
SDN	Software Defined Network
SenML	Sensor Markup Language
SMS	Short Message Service
TCP	Transmission Control Protocol
TKL	Token Length
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WAN	Wide Area Network
WLAN	Wireless Local Area Network
WoT	Web of Things
WPAN	Wireless Personal Area Network
WSN	Wireless Sensor Network
WWW	World Wide Web

Contents

Preface i

Glossary ii

1 Introduction 1

1.1 Research problem 3

1.2 Research process 3

2 Internet of Things architecture 4

2.1 Machine to machine 4

2.1.1 ETSI M2M 5

2.1.2 oneM2M 6

2.2 RESTful architecture 7

2.2.1 HTTP 7

2.2.2 Constrained Application Protocol 8

2.3 Publish-subscribe paradigm 13

2.3.1 MQTT protocol 13

2.4 Web of Things 15

2.4.1 Evolution of the Web towards the Web of Things 16

2.5 IoT components 17

2.5.1 Things 18

2.5.2 Gateways 19

2.5.3 Back end 19

2.5.4 Applications 19

2.5.5 Interfaces 20

3 IoT connectivity 21

3.1 Internet reference models 21

3.1.1 Application layer 22

3.1.2 Transport layer 23

3.1.3 Network layer 23

3.1.4 Data-link layer 23
3.1.5 Physical layer 23
3.2 IoT protocols and protocol stacks 24
3.2.1 IEEE 802.15.4 25
3.2.2 IEEE 802.11 27
3.2.3 Bluetooth 27
3.3 Future connectivity 29

4 IoT gateway **31**
4.1 Gateways and routers 32
4.2 Gateway functionalities 33
4.2.1 Edge computing 33
4.2.2 Dataflow control 33
4.2.3 Proxy functionality and caching 34
4.2.4 Resource discovery 35
4.2.5 Security 36
4.3 Gateway management 36
4.3.1 Management standards 36
4.3.2 Management protocols 38
4.4 Virtualising the gateway functionality 38

5 IoT gateway development and deployment **40**
5.1 Gateway architecture 41
5.1.1 Distributed Decision Engine 41
5.1.2 Data format 42
5.1.3 Hardware interfaces 43
5.1.4 Operating system 44
5.2 Interface design 44
5.2.1 Wi-Fi Producer 45
5.2.2 XBee Producer 46
5.2.3 BLE Producer 47
5.2.4 CoAP Consumer 47
5.2.5 HTTP Consumer 47
5.2.6 MQTT Consumer 48
5.3 Case examples 48
5.3.1 Case 1 - Environmental monitoring 51

5.3.2 Case 2 - Tracking an object/indoor localisation 52
5.3.3 Case 3 - Crowd counting 53
5.4 Discussion 53

6 Conclusion **56**

References **58**

1 Introduction

The Internet of Things (IoT) has been a widely discussed topic for a while now. However, it is not always so obvious as to what is meant by that term. The name was first used by Kevin Ashton in 1999 [6]. He viewed it as an extension of the traditional Internet, where not only humans generate data, but different kinds of objects do so as well.

Atzori et al. [7] view the IoT as an intersection between semantic, Internet and things oriented visions. The Internet oriented IoT vision emphasises the connectivity of IoT devices, while the IoT devices cover the things oriented vision. The semantic oriented vision includes organising, representing, storing, searching and interconnecting information.

Vermesan et al. [78] consider the IoT to form a greater internet together with the Internet of Energy, Internet of Media, Internet of Services and Internet of People. Cisco [27] views the IoT as physical items, which together with people, data and processes form a "network of networks" called the Internet of Everything (IoE). The IoE would benefit individuals, businesses and governments. In Cisco's vision, the IoE can help with tracking world resources and solving major problems, such as hunger, limited drinkable water resources and climate change.

The IoT covers a very heterogeneous set of devices, which operate in various environments. IoT devices can be found in household appliances, wireless sensor networks (WSNs), industrial machines, jet engines and so on. The point is that they can be put everywhere, and that creates many new challenges. Chase [16] lists connectivity, power management, security, complexity and rapid deployment as challenges of the IoT. Gubbi et al. [30] append the list with privacy, participatory sensing, data analytics, geographic information system (GIS) based visualisation and cloud computing. They also consider architecture and protocols as open challenges, which also play key roles in solving IoT connectivity problems. Atzori et al. [7] consider standardisation activity and networking issues as open challenges in the IoT. The open challenges considered by Miorandi et al. [55] include distributed intelligence, distributed systems, computing, communication and identification. It is worth noting that connectivity is present in every list in the aforementioned open research challenges of the IoT, in one form or another.

Machine to machine (M2M) is an integral part of the IoT. M2M consists of autonomous devices communicating with each other, without human intervention. M2M communication does not necessarily require Internet connectivity. In fact, if M2M systems are connected to the Internet, we can view the resulting system as an IoT system [14].

The traditional Internet is powered by the TCP/IP reference model. TCP/IP reference models have a layered architecture with four layers. Internet protocol (IP) addressing is used for routing packets to their destinations. The TCP/IP model is also relevant to IoT systems, but in a much lighter form than with traditional networks. Another popular reference model is the OSI model, which has seven layers. However, nowadays, the OSI model mostly serves as an educational model rather than a design specification [77], [50].

IoT devices can be constrained in several properties: the common ones being energy consumption and storage, physical size, cost, computing power and memory. These factors also set constraints on connectivity. Devices may not be able to send data frequently and their communication range is limited. Devices also come with various radios and protocols, making the connectivity problem even more complicated. The goal is to gather data from these devices, deliver them to the Internet and make useful applications around them. Often a bidirectional communication is preferred, as we want to manage the end devices with control messages. In this thesis, the connectivity problem is approached by designing a flexible IoT gateway that can host several radio interfaces and is easily modifiable, for new use cases.

An early version of the IoT gateway was developed for providing connectivity to a safety vest, which was equipped with sensors, a GPS module and a small IEEE 802.15.4 radio [43], [44]. The sensor vest was later equipped with wireless charging capabilities, in a follow-up research project [45]. The gateway, however, was not upgraded, at that point. Soon, several other research and commercial projects required a gateway for their specific purposes. It often meant that the gateway had to be tailored for each project individually. That led to an idea of a generic IoT gateway that would be easy to deploy with new scenarios. The empirical part of this thesis presents an IoT gateway design as well as three case examples, where the gateway is used.

The rest of the thesis is organised as follows: Chapter 2 gives an overview of the components and their relations that, together, form the IoT architecture. Also, standards related to IoT architecture are briefly discussed. The Internet reference models and the IoT protocol stacks are discussed in Chapter 3. Chapter 4 focuses on the IoT gateway and discusses its requirements and functionalities. IoT gateway management is also touched in this chapter. Chapter 5 covers the empirical part of the thesis, presenting the design and deployment of an IoT gateway. Three cases are presented as example usage scenarios for the gateway. In Chapter 6, a summary of the topics covered in this thesis is given and conclusions, based on the theoretical and empirical parts, are drawn.

1.1 Research problem

As noted in [16], [30] and [7], the connectivity of the IoT devices is an open challenge that needs to be solved using new protocols, architectures and standards that are specifically tailored for the IoT. This thesis takes a gateway oriented approach for the connectivity issue. The main questions are: how to provide connectivity using an IoT gateway, what are the main functionalities of an IoT gateway and how well does a gateway fit into the IoT architecture, in contrast with other options.

1.2 Research process

This thesis was done at VTT as part of the Digile IoT programme, the VTT-funded Energy Aware Learning in Cognitive Radios and Networks (AWARENESS) project and the Tekes-funded 5G Test Network (5GTN) project. The beginning of the gateway research and development dates back to 2014, when a preliminary version of the gateway was introduced in [44]. Gateway development continued in [43], where it was used as part of an indoor localisation system. In [45], the gateway was applied to a larger use case, but its functionality was kept the same as in the previously mentioned publications. The gateway was an essential part of the scenarios presented in the publications, but it was never the focus of research. This thesis builds on the work that was previously done and focuses on the gateway topics. The goal of the empirical part is to design a fully functioning IoT gateway and form a gateway infrastructure that would serve as a testbed for future research, for example, in energy-efficient communication protocols and localisation algorithms. The author's role in this process was to define use cases, develop gateway software and deploy a network of gateways in a laboratory environment.

2 Internet of Things architecture

This chapter gives a high level picture of the IoT field. The machine to machine (M2M) standardisation efforts of ETSI and oneM2M are briefly discussed in separate sections. IoT and M2M have subtle differences, but they can often be considered to mean the same thing. Similarly, the M2M specifications presented in this chapter can be viewed as the IoT specifications. RESTful architectural style is introduced and Constrained Application Protocol (CoAP) is more carefully inspected. The publish-subscribe paradigm is discussed as it occurs in many places in the IoT architecture, as will be seen later in the empirical part. The concept of the Web of Things is introduced as it can also be viewed as a platform for new IoT applications. Lastly, IoT components that, together, form the IoT architecture, are discussed.

2.1 Machine to machine

Machine to machine (M2M) communications form an integral part of the IoT. In [14], Boswarthick et al. consider M2M as communication between an end device and a business application. However, the communication between the two has to be operated with little or no human intervention in order to be qualified as M2M. M2M, itself, is not a standard. Rather, it is a paradigm in which devices autonomously communicate with each other. In the literature, M2M is sometimes referred to as machine type communication (MTC). The term is most commonly found in 3rd Generation Partnership Project (3GPP) specifications [76].

Due to the heterogeneous set of technologies involved in M2M, the solutions often turn into so-called "silos". Boswarthick et al. [14] view standardisation and reusable service platforms as a means to take the silos down and increase the interoperability in M2M. This is often referred to as a transition from vertical silos to horizontal platforms. Alam et al. [4] consider the future IoT to offer a horizontal platform for a multitude of vertical M2M applications. They also view standardisation as being one of the key elements to move M2M and the IoT forward. ETSI M2M and oneM2M architectural standards are briefly presented in the following subsections. In addition, IEEE has set up a working group for specifying an IoT architecture, but as of the writing of this thesis, they have not yet published a standard, and thus, it is not covered in this thesis [35].

2.1.1 ETSI M2M

The European Telecommunications Standards Institute (ETSI) has set up a technical committee working on standardising the M2M architecture. The ETSI M2M standard supports short-range communication, but the emphasis is on long-range cellular communication, which allows mobility and provisioning. Considerable fragmentation exists in the form of component level standards tackling, for example, how a sensor node communicates with a gateway. ETSI M2M aims to offer a standardised way to bring the fragmented application areas onto a horizontal platform. The standardised platform allows the M2M market to takeoff and deliver cost-effective M2M solutions. Fig. 2.1 depicts the ETSI M2M high-level architecture.

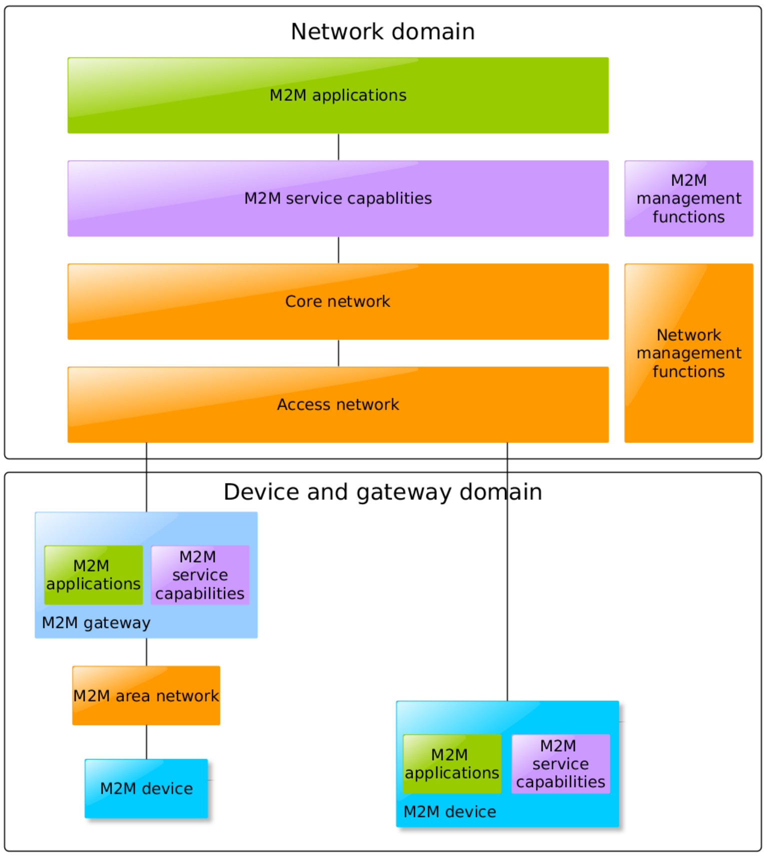


Figure 2.1: ETSI M2M high-level architecture. Adapted from ETSI M2M functional architecture document [25].

The high-level architecture is separated into two domains: network domain and device and gateway domain. There are two types of end devices in the device and gateway domain. The first type of M2M end devices operate in constrained M2M area networks and require an M2M gateway for accessing the network domain. M2M area networks can be mesh-networks or simply point-to-point connections between an M2M device and a gateway. The second type of devices are capable of communicating with the network domain directly, without the need of gateways. The core network at least has IP connectivity, and the M2M devices connect to it through access network base stations. M2M service capabilities offer M2M functions that M2M applications access, using open interfaces. M2M management functions are used to manage the M2M service capabilities. The core network and access network are managed using network management functions. [25]

2.1.2 oneM2M

oneM2M [59] is a global standardisation body for M2M and the IoT. It was formed in 2012 by seven major standards organisations. The goal of oneM2M is to standardise interoperability between devices, applications, data collection and data storage. It aims to enable growth in M2M and IoT markets by providing a set of specifications for an IoT architecture [59]. oneM2M is intended to work with existing industry standards. It can also be viewed as a partial extension of the ETSI M2M standard [25]. The oneM2M functional architecture document [60] specifies a service layer, which aims to provide a single network-independent horizontal platform for the multitude of the IoT domains. oneM2M does not specify specific technologies or protocols to be used. Instead, it specifies higher level interfaces, which businesses use to access services. Therefore, in a high level scenario, oneM2M can be viewed as providing interoperability between businesses, through a common service layer. Other important aspects of oneM2M are security and device management. Device management is enabled by integrating existing device management standards into oneM2M specifications. Device management and management standards are discussed in Chapter 4.

2.2 RESTful architecture

Representational state transfer (REST) is an architectural style used in the World Wide Web (WWW). RESTful systems work in a request-response manner. A client makes a request to a server and the server sends a response to the client. The server specifies a set of interfaces that the client can use for accessing and modifying resources. The message sent by the client contains a request method, which tells what kind of operation is requested to be done on the server resource. The server sends back a response containing a code, which tells whether the request was successful or if there was an error. RESTful systems typically use HTTP in the Web, but it is often unsuitable for IoT applications. The Constrained Application Protocol (CoAP) has been developed to provide RESTful style for wireless sensor networks (WSNs) and the IoT as well as the Web of Things (WoT), which will be discussed in Section 2.4.

2.2.1 HTTP

Before diving into the mysteries of CoAP, it is worth having a short recap of the HTTP protocol, as there are many similarities between the two. HTTP [28] is a popular application layer protocol used in the Web. The most common HTTP request methods are GET, PUT, POST and DELETE. GET returns a representation of a Web resource to the user. The requested resource is specified by the Uniform Resource Locator (URL), which includes the path to the server and to the requested resource. The GET request typically yields a HyperText Markup Language (HTML) document as a response, but it could also be, for example, an Extensible Markup Language (XML) file or a JavaScript Object Notation (JSON) file. POST and PUT methods are used to create or modify server resources. The DELETE method removes a given resource. HTTP communication typically uses TCP as the transport protocol, as reliable transmission of packets is often required. The IoT can benefit from the HTTP protocol, especially when it is used between devices that have plenty of processing resources. Unfortunately, those devices are often only mains powered gateways or back end devices, because HTTP is typically too much of a resource demanding protocol to be implemented in an IoT end device.

2.2.2 Constrained Application Protocol

Constrained Application Protocol (CoAP) [72] was developed to address the limitations that other application level protocols, such as HTTP, have in low-power and lossy networks (LLNs). CoAP provides a common language for the heterogeneous IoT applications and aids the emergence of the IoT as well as the Web of Things (WoT). CoAP is currently a trending topic in the IoT field. A search on the IEEE Xplore digital library yields 912 results with a search string "'Constrained Application Protocol' OR CoAP". The search covered metadata and full text. Filtering the search to the years 2014–2015, yields 513 results, so it clearly shows that a lot has been going on around CoAP, lately. Of course, not all the papers focus on developing or enhancing CoAP. Many papers merely build applications or test other concepts, with the help of CoAP. It still indicates that CoAP has reached a certain level of maturity, and can be used in many different scenarios.

CoAP resembles HTTP in many ways, but it is not merely a reduced or compressed version of the HTTP protocol. Like HTTP, it uses GET, POST, PUT and DELETE methods for querying and modifying resources. CoAP has new features that are especially designed for the constrained environments and are not part of the HTTP. One of them is the Observe method [34], which allows a client to continuously receive responses from a server. It is a useful method, for example, in a case where the client wants to receive a continuous stream of sensor data from a sensor node. Energy is saved due to the reduced amount of control messaging. CoAP supports resource discovery, i.e. the protocol allows the client to request information on the resources that the server has. Secure transmissions at the transport level can be achieved by using a Datagram Transport Layer Security (DTLS) protocol. CoAP supports HTTP proxies, meaning that clients can request resources from a CoAP server using regular HTTP requests. The CoAP protocol can be further optimised by using header compression techniques, as has been done in [54].

CoAP messages are sent using the User Datagram Protocol (UDP). UDP reduces the amount of control messaging, as it does not guarantee reliable delivery like the Transmission Control Protocol (TCP). However, CoAP can provide reliable delivery on the application layer. The sender can tag messages as confirmable, which requires an acknowledgement from the receiving end. The sender repeats sending the confirmable messages until it runs out of attempts or the recipient acknowledges it with an acknowledgement message. The number of attempts is defined by the MAX_RETRANSMIT variable.

Message types

CoAP specifies the following message types:

- Confirmable
- Non-confirmable
- Acknowledgement
- Reset

As already mentioned, confirmable messages require acknowledgements and they are used when a reliable delivery is required. The sent confirmable messages are matched to the received acknowledgement messages, by using message IDs. Non-confirmable messages must not be acknowledged. Non-confirmable messages suit well in situations where streams of data are sent. That also means that the messages may be lost during transmission or arrive out of order. CoAP also elegantly supports piggybacked messages. A recipient of a confirmable message may send payload data back on an acknowledgement message, instead of building a new confirmable message, as the response. If the recipient cannot immediately respond with a piggybacked acknowledgement message, it will send an empty acknowledgement and later send a confirmable or non-confirmable message with the payload content in it. The reset message is sent when the receiver cannot handle the received message, for one reason or another. In general, reset messages are used to reject unknown messages.

Message format

A CoAP message is composed of a header, a token, a set of options and a payload. The structure of a CoAP message is depicted in Fig. 2.2.

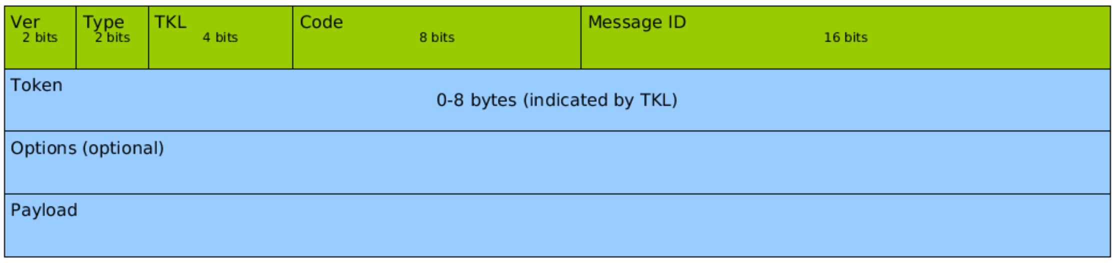


Figure 2.2: CoAP message format. The header is highlighted in green.

The message header has a fixed length of 4 bytes. The version field indicates the version of the CoAP specification. Value 1 means that the current RFC-7252 CoAP specification is used. Other values are reserved for future versions. The next field in the header is the type field. It indicates whether the message type is confirmable, non-confirmable, an acknowledgement or reset. The token length (TKL) field contains the length of the Token, which can be between 0 and 8 bytes. The code field indicates the request method code, in the case of a request message or the response code, in the case of a response message.

The 0-8 bytes long token field is used to match requests to responses. It is a different concept from the message ID, which only matches individual CoAP messages. Tokens are allowed to be up to 8 bytes long to protect against spoofing attacks. All CoAP messages have tokens, even if they have zero length.

The options field holds information that affects the performance and functionality of the CoAP. For example, a remote resource is identified by a Uniform Resource Indicator (URI). The URI is decomposed into the option field in the CoAP message, when a client makes a request. The format of the CoAP URI scheme is shown below:

coap-URI = "coap:" "//" host [":" port] path-abempty ["?" query]

The above example is taken from the RFC-7252 [72]. The host section can contain a host name or an IP address. The port, used by UDP, points to the application running on the server. The default port 5683 is used if the port section is left empty. Furthermore, the URI contains a path to the resource and a sequence of parameters for more detailed queries.

Caching and proxying related information are also given as options. An exhaustive list of options can be found in the RFC-7252 [72]. The payload data can be contained in a request or a response message. Payload can be carried in a confirmable or a non-confirmable message, or it can be piggybacked on an acknowledgement message. The content-type in the options field indicates the type of payload. If the content-type is not specified, then it can be inferred from the application context.

Requests

The basic CoAP request methods are GET, POST, PUT and DELETE. The GET method is used to request the state of a resource that is given in the URI. It can be a sensor value, battery status, name of the device and so on. The PUT and POST methods are used to update the value of a resource or to create a new resource. The DELETE method is used to remove the resource specified by the URI.

In addition to the basic RESTful request methods, the RFC-7641 [34] introduces an Observe method for the CoAP protocol. It was designed because the existing methods did not work well when a client was interested in observing a resource, over a period of time. The protocol allows a CoAP server node to send notifications continuously, after it has received a registration message from a client. The server keeps a list of all registered observers. The server's goal is to keep the client up-to-date by notifying the observers of the latest values. The observed phenomenon can be sent to the client at regular intervals or when a change in the value has occurred. It is up to the server to decide the conditions of when to notify the client.

The method is based on the observer design pattern, which is well-known in the software engineering discipline. The messages carry observer-related information in the options field. When a client is interested in observing a resource, it sends a registration message to the server. The registration message is sent as a GET method, with the observe option set to value ″0″. The server adds the observer to the observer list and starts sending notifications. Each notification message has a value set to the observe field, and it is used to check the freshness of the measurement. If the server is not able to add a new observer, it sends a response without the observe option. The basic operation of the observe protocol is depicted in Fig. 2.3. The client is interested in observing the temperature at the server node and starts with sending a registration message to the server. The server adds an observer to its database and starts sending notifications to the client. When the client is no longer interested in observing the temperature, it sends a deregister message with an observe option set to ″1″.

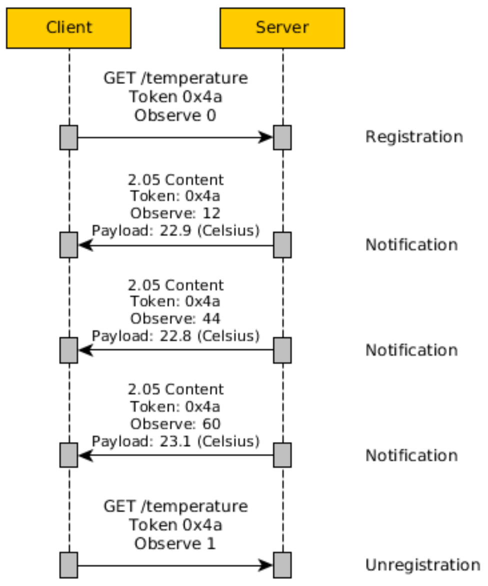


Figure 2.3: Sequence diagram of the CoAP observe protocol. Adapted from RFC-7641 [34].

The server can send the notifications in confirmable or non-confirmable messages. In the case of confirmable messages, the server awaits acknowledgements from the client. If the server does not receive acknowledgements within a predefined period of time, it will consider that the client is no longer interested in observing the resource. The server will then remove the client from the observer list. This is the easiest way for the client to stop observing a resource. There are two alternative ways for the client to stop observing. The client can deregister by setting the observer option to ”1”, as depicted in Fig. 2.3, or it can reject a notification by sending a reset message. In both cases, the server will remove the observer from the list.

Responses

If the request method was delivered in a confirmable message, it should yield an acknowledgement message containing one of the following response code classes:

- 2.xx Success
- 4.xx Client error

- 5.xx Server error

The ’xx’ values are replaced with the actual codes that more precisely identify the response code. If the resource was successfully retrieved, updated, created or deleted, then it should yield a success response. Client errors occur when the sender is assumed to have made a mistake. Incorrect URI’s, requesting non-existent resources or resources to which the sender does not have access, should all yield client errors. Server errors happen when the server cannot handle the request or is faulty, for some other reason.

2.3 Publish-subscribe paradigm

The publish-subscribe paradigm [26] makes up another high-level architectural style for message transmission. The paradigm is roughly composed of publishers, an event service and subscribers. Publishers produce events that are passed to the event service. Events are then forwarded to the subscribers who can then consume them. The publish-subscribe paradigm has a fundamental problem as the structure of the published data has to be known by the subscribers. Also, publishers should not arbitrarily change the structure of data, as it may break the subscribing modules. In larger and more complex systems, this may lead to very messy implementations. However, the publish-subscribe paradigm is an effective way of distributing information from publishers to subscribers, and is very useful in IoT applications. The IoT friendly MQ Telemetry Transport (MQTT) is a good example of a protocol using the publish-subscribe paradigm.

2.3.1 MQTT protocol

MQ Telemetry Transport (MQTT) [10] is a well-known application layer protocol suitable for low-power and lossy networks. The publish-subscribe implementation is based on the TCP protocol. Clients publish messages via a centralised broker, which forwards them to subscribers. Messages are published as topics that can be, for example, temperature, acceleration or light switch status. Subscribers can subscribe to a topic or a set of topics using topic filters and wild cards.

Messaging

MQTT defines five different messaging methods: connect, disconnect, subscribe, unsubscribe and publish. The messages are carried in MQTT control packets. Connect is used by subscribers to establish a connection to a broker. Subscribers close the connection by using disconnect method respectively. Connection is needed before any subscriptions can be made. However, publishers do not need to set up a connection beforehand, in order to start publishing a topic. Acknowledgement messages are specified for connect, subscribe, unsubscribe and publish messages.

An MQTT control packet is composed of a fixed header, a variable header and a payload. A fixed header is part of every control packet. It contains fields for the packet type, flags and remaining length. The presence of the variable header and payload depends on the message type. A packet identifier field is commonly found in variable headers of publish, subscribe and unsubscribe messages. The packet identifier is used for retransmissions when quality of service is enabled. The content of the payload field also varies, depending on the message type. In case of a publish message, the payload contains the actual MQTT application message.

Topics

Clients either publish or subscribe to a topic. In sensor networks the topic could be, for example, temperature in a specific room or the state of a light switch. However, topics are not restricted to sensors, and payloads are not restricted to numeric values. In fact, topics can be anything and the published messages may contain data in many different forms. Subscribers naturally need to know what to subscribe and, therefore, MQTT specifies topic filters and wildcards to enable more advanced subscription functionality. The topic is composed of topic separators and topic levels.

For example, ”/home/kitchen/temperature” has three levels, which are separated by slashes. In the example, several different sensors can be located in the kitchen and can be published as their own topics. Wildcards can be used to specify a set of topics of interest. A ”+” character is used to match a single topic level. A topic string ”/home/kitchen/+” would return messages from all sensors located in the kitchen. On the other hand, a topic string ”/home/+/temperature” would return temperatures of all rooms in the home. A ”#” character is used to match all the levels in the path. A string ”/home/#” would match all of the sensors in all rooms including, e.g. "/home/livingroom/humidity" and "/home/kitchen/temperature".

Comparison to CoAP

Even though CoAP achieves similar behaviour with the Observe protocol, the two protocols are fundamentally different. CoAP connections are typically formed between two nodes, whereas MQTT forms connections between multiple clients. There may be many subscribers for a topic and many clients publishing on the same topic. The CoAP Observe protocol uses a server to generate and publish messages to subscribers. There are no external publishers who would first publish messages to the CoAP server. However, there is work in progress to provide similar functionalities to CoAP by introducing a publish–subscribe broker [48]. The basic idea is the same as with MQTT. CoAP clients publish messages on a topic via a centralised CoAP server that operates as a broker. The broker further forwards messages to matching subscribers.

2.4 Web of Things

M2M and IoT standardisation were discussed in Section 2.1. The standardisation efforts were aiming to offer a common platform for the heterogeneous set of M2M and IoT applications. In this thesis, the term Web is often used to shorten World Wide Web (WWW). The World Wide Web has been successful in providing a platform for the multitude of applications running on the Internet. There is a great deal of research being done to apply the same principles to the IoT. The IoT version of the Web is commonly known as the Web of Things (WoT). The Web of Things is not a separate concept from the Web. Rather, the WoT is extending the Web with Things by using similar protocols and design principles. Guinard and Trifa [31] consider that it is worth leveraging existing Web protocols, as they can also provide interoperability between the real objects. Mainetti et al. [53] also see the Web as having the potential to operate as the common platform for the heterogeneous set of IoT applications, but some of the protocols need to be optimised.

Applications built on top of the WoT are called mashups [31]. Mashups gather data from different sources, e.g. physical devices in order to provide some sort of a service. One of the biggest challenges of the WoT is accessing these devices. The physical devices operate as servers and serve their resources the same way Web servers offer their resources. The difference is that Web servers have plenty of processing and storage capacity, whereas embedded servers have only a tiny fraction of it. Moreover, a large amount of traffic to and from the embedded server will deplete its battery very quickly. One solution to this problem is explained in Section 4.2.3, which discusses proxying and caching.

HTTP has proven to be a powerful protocol for building Web applications and Guinard and Trifa [31] have used it for demonstrating WoT mashups. One of the characteristics of the WoT is that the underlying architecture is hidden from the end user who is accessing the resources using, for example, a Web browser. The use of proxies enables translation between protocols allowing the creation of many different kinds of architectures. A user may use a regular Web browser and make requests in HTTP to access IoT resources. The user may or may not know that there is a proxy translating the request to a protocol that the IoT device understands. There even exist Web browser extensions, such as Copper [49], which enables full CoAP-to-CoAP communication between a Web browser and a CoAP server.

2.4.1 Evolution of the Web towards the Web of Things

As computers have been getting more powerful in terms of processing capabilities, they have also been getting smaller and cheaper. Embedded devices have been able to host simple HTTP based Web services, for some time. An example of this could be a wireless access point, which typically has an embedded Web server running inside to serve a configuration page using HTTP. Optimising the traditional communication protocols and inventing new ones has made it possible to use even smaller and more constrained embedded devices for IoT and WoT purposes.

The Internet was a very disorganised and modest network before the introduction of the World Wide Web, in the early 90s. The Web has made new innovations possible and unleashed the real power of networked computers and information sharing. The Web has enabled people to easily generate and store information, as well as find that information using search engines. In [1], this first revolutionising wave of the Web is referred to as Web 1.0. The Web 2.0 strengthened collaboration between individuals and enabled collaborative platforms such as wikis [1]. Social media took off and offered even higher layers for new innovative applications. Now there is even active research on combining principles from the social networks and the IoT [8]. People are generating a large volume of data on the Web, and that requires intelligent information processing algorithms and storage techniques. The amount of data generated on the Web is increasing even more rapidly, as more and more Things are connected to the Internet. This clearly shows that the Web has come a long way, in a very short time; but, is still thriving and always finding new areas for growth.

2.5 IoT components

Fig. 2.4 gives a simplified overview of the components, which form the IoT architecture. The idea behind the figure is that there is a set of Internet and IoT components that, together, form a core for the IoT applications. In the figure, the IoT applications are placed around the core components. There are certainly many more devices and applications involved, but, the main point is to give a basic idea of what the IoT is.

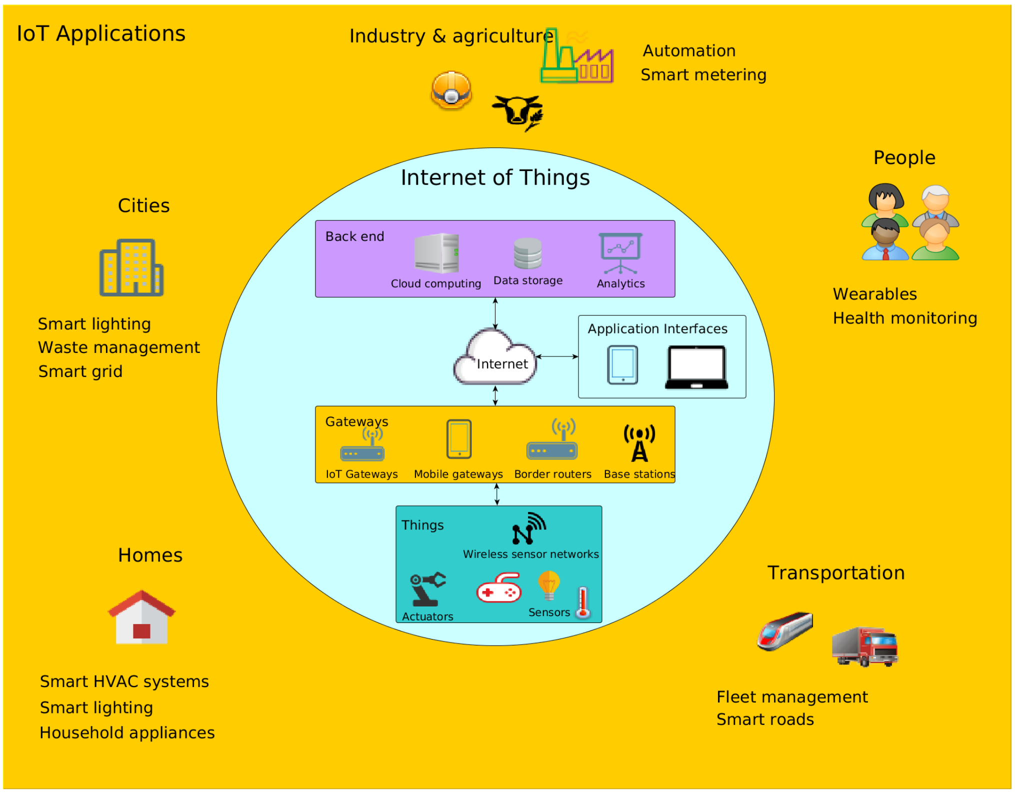


Figure 2.4: A general view of IoT architecture and application areas

Categorising applications to different domains is a bit of a fuzzy activity and there are many different ways of doing it. There is also a lot of overlap between the different application domains. This section explains the different IoT components, which include end devices, network devices, the back end and interfaces. Lastly, some of the applications and domains are discussed.

2.5.1 Things

Things or IoT devices are commonly categorised as sensors or actuators. They are roughly composed of a power supply, a radio and a microcontroller. Sensors are used to gather information from the surroundings. The sensed observables can be, for example, temperature, various gases, humidity and movement. Actuators, on the other hand, can interact with the environment, e.g. by showing information on a monitor, flashing an LED or moving a robot arm. Sensor readings can be used as inputs to actuators. For example, an actuator can optimally set the room lighting if given the brightness readings, or an actuator can adjust the radiator if given the temperature values of the environment.

IoT devices are also minimal on the software side. Generally, they are not able to run Linux sized operating systems. There exist several lightweight operating systems intended for small sensor nodes. Two of the well-known ones are Contiki and TinyOS. Sensor node operating systems are not in the scope of this thesis and an interested reader is encouraged to see a survey by Hahm et al. [32] for the current state of the art.

IoT devices are often not mains powered, and require a battery and/or energy harvesting techniques. It is not practical to change batteries often, as there may be hundreds or thousands of IoT devices in the network. Therefore, the power consumption of the device has to be very low in order to increase the lifetime of the device. Additionally, different energy harvesting techniques can be used to recharge the battery on the go or to immediately initiate transmission. Energy sources include, for example, light, wind, human movement, ambient vibrations and RF energy [75].

Radio-frequency identification (RFID) technology introduces another interesting way of gathering energy for information processing and transmission. Passive RFID tags get their energy from RF signals coming from the RFID reader. Therefore, they do not require batteries. Active RFID tags, however, have their own power supplies, and can initiate communication [79].

2.5.2 Gateways

Gateways process and relay information between the IoT devices and the Internet. Gateways are needed as IoT devices are not typically able to connect directly to the Internet. Gateways come in several different forms. Gateways include mobile phones, specialised IoT gateways and also 4G/LTE base stations that can be thought of as gateways. Sometimes IPv6 connectivity is needed in the end device nodes. Connectivity is then enabled by border routers, which implement a network level protocol stack. Gateways are discussed in greater detail in Chapter 4.

2.5.3 Back end

Resource demanding operations, such as information processing and data storage, are often done in the IoT back end. Data coming from the IoT devices are stored in databases, which are located in the cloud. The volume of the data coming from the IoT devices is often massive and can be considered as big data. Data mining, machine learning and analytics algorithms can be run in the back end to find new information from raw data. The back end offers services that can be used by users and/or devices to request information. A Web server is a common example of a service running in the back end.

2.5.4 Applications

There are various ways of categorising IoT applications into domains. Atzori et al. categorise applications into transportation and logistics, healthcare, smart environments, and personal and social domains [7]. Gubbi et al. use four categories: personal and home, enterprise, utilities, and mobile domain [30]. There is a lot of overlap between the categories, as shown in Fig. 2.4, which gives some examples of the applications and application domains. The point is that the IoT touches many different domains and can be applied to almost anything.

The IoT is becoming more popular in industry and the term industrial IoT has been coined to refer to this subset. Xu et al. [84] list healthcare services, food supply chain, mining production, transport and logistics, and firefighting as some of the application domains for the industrial IoT. The industrial IoT enables more efficient use of resources by applying methods from artificial intelligence. Alahakoon and Yu [3] consider smart meters and smart grids as the ”way of life” in the future.

Smart meters can monitor energy usage in homes so that electricity consumption can be adjusted accordingly. The energy sector benefits from the industrial IoT, in the form of smart grids. A smart grid enables more efficient management of the grid and allows better communication between the electric industry and homes. Smart metering and smart grids not only allow homes to adjust their energy consumption, but also the electric power industry to adjust the electricity generation by demand, at given times. Different energy sources can be more easily added to a smart grid, including devices that locally generate electricity at homes.

2.5.5 Interfaces

In Fig. 2.4, the two-way arrows indicate connections between different sections of the IoT. Many kinds of Things interface with the physical world by observing and manipulating it with sensors and actuators. Things communicate with gateways through physical wireless or wired connections. Gateways have similar methods for interfacing with the Internet infrastructure. However, IoT applications require richer communication than just a physical connection to exchange data. Higher level protocols enable more advanced functionality and features for M2M communication. IoT applications can talk to each other using, e.g. RESTful interfaces that were presented in Section 2.2. Section 3 discusses IoT connectivity in more detail, by introducing Internet reference models as well as protocol stacks for the IoT. IoT standards, such as oneM2M, provide even wider integration by introducing service layer interfaces that the different application domains can use to communicate with each other.

Finally, the IoT is not just devices or machines communicating together. There has to be a way for humans to access data that is gathered or generated by the devices. This type of communication is known as Human–computer interaction (HCI) [65]. HCI is enabled through user interfaces, which can be, for example, graphical or text-based computer programs, mechanical switches, or voice controlled interfaces. In order to provide the greatest value for the user, the raw data is processed and refined into human readable format. Visualisation of the data is a good technique for representing complex data. Two-way communication between devices and users is often required. The typical cases include navigating resources and querying data. There may also be management tasks for controlling the IoT devices and gateways directly through the user interface.

3 IoT connectivity

As mentioned before, the IoT does not form a new Internet. Instead it enhances the existing Internet by extending it with "Things". Therefore, it is important to review the existing Internet protocols and examine their applicability in the IoT. This chapter takes a look into the Internet reference models as well as the protocol stacks used in the IoT.

3.1 Internet reference models

This section presents three models that are used to describe Internet connectivity or are used as a guideline for designing network devices and software. The models are the OSI model, the TCP/IP model and their hybrid model. The models are depicted in Fig. 3.1.

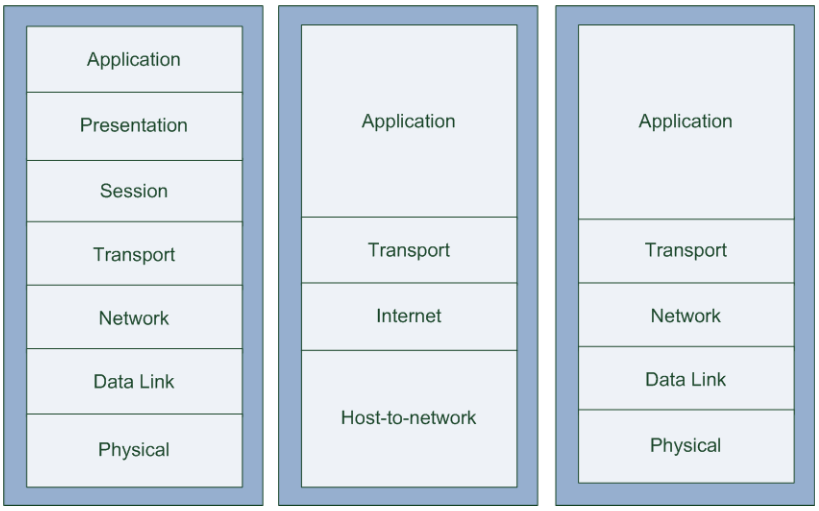


Figure 3.1: From left to right: OSI model, TCP/IP model and a hybrid model

As mentioned in the introduction, the TCP/IP reference model is the basis of the connectivity in the traditional Internet. It has a layered architecture of four layers. They are from the highest to the lowest: application layer, transport layer, Internet layer and host-to-network layer. In the layered architecture, lower level layers offer services to higher level layers. Services include, e.g. encapsulation, encryption and retransmission. For example, the transport layer encapsulates the application layer data into datagrams. The Internet layer takes the transport layer datagrams and encapsulates them into packets and gives them Internet protocol (IP) addresses. Finally, the host-to-network layer encapsulates the packets into frames and sends them through a wired or a wireless link to the next hop.

According to Tanenbaum [77], the TCP/IP reference model was developed for the ARPANET, which was the predecessor of today's Internet. The name of the model came from the two fundamental protocols used in the ARPANET. Today, the TCP/IP protocols are still the cornerstones of the Internet connectivity. The protocol stack has since been appended with newer protocols that are used in a wide variety of scenarios.

The OSI model is another well-known networking reference model. It defines seven layers. In addition to the TCP/IP model's layers, it contains presentation and session layers. Also, instead of a host-to-network layer, the OSI model has a data-link layer and a physical layer. The OSI reference model is rarely used as the basis for design. It mostly serves as an educational model for network connectivity.

In the literature [77], [50], a hybrid model inspired by the TCP/IP and OSI models is often used. In the hybrid model, the TCP/IP model's host-to-network layer is further split into data-link and physical layers. The OSI model's session and presentation layers are abandoned, as they are rarely needed and they can be implemented in the application layer if required. The resulting reference model, therefore, has five layers: application, transport, network, data-link and physical.

3.1.1 Application layer

The application layer is the highest level in the reference models. Network applications can be viewed as interfaces between the user and the network. The Web browser is a familiar application that uses an application layer protocol, namely HyperText Transfer Protocol (HTTP), for communicating with a Web server. Other important protocols include File Transfer Protocol (FTP) and Simple Mail Transfer Protocol (SMTP).

3.1.2 Transport layer

The transport layer handles the connections between applications over a network. Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are the two common, currently used transport layer protocols. TCP provides reliability, ordering of packets and error checking for the communication. If such reliability is not needed, then connectionless UDP is a good alternative. UDP is suitable for applications that do not care if some packets are lost during transmission or occasionally arrive in the wrong order.

3.1.3 Network layer

The network layer encapsulates datagrams from the transport layer into packets. Packets are delivered to their destinations using IP addressing. IPv4 is widely used in the Internet, but its address space has already been exhausted. Therefore, IPv6 has been specified to enable the Internet to keep up with the growing number of devices. Its success is also critical for the success of the IoT. IPv6 addresses are 128 bits, meaning that the address space has around 10^{38} addresses [7]. To put things into perspective, Mulligan calculated in [56] that there are 667×10^{21} IPv6 addresses for every square meter on Earth. Therefore, there are plenty of addresses for the future IoT devices, as well.

3.1.4 Data-link layer

The data-link layer handles communications between adjacent devices. Its responsibilities include encapsulating packets into frames, accessing the link using a media access control (MAC) protocol, providing reliable delivery of frames, and detecting and correcting errors. The Ethernet and 802.11 (WLAN) are two examples of MAC protocols used in the data-link layer. A common IoT example is the IEEE 802.15.4 standard, which was specified for low-power embedded devices. It forms the basis for many mesh protocols that are used in IoT and sensor networks.

3.1.5 Physical layer

Lastly, the physical layer (PHY) takes the data-link layer frames, encodes them into individual bits and sends them over a communication link. Similarly, incoming bits of information are decoded and then passed onto the data-link layer. The link can be wired or wireless. In the case of a wireless link, it is the responsibility of the physical layer to set the frequency for transmission and reception [77],[50].

3.2 IoT protocols and protocol stacks

This section presents protocols and protocol stacks that are especially used between an end device and a gateway. Whilst Internet devices typically use the TCP/IP protocol stack, IoT devices have many to choose from. In [71], Shelby and Bormann presented an IPv6 based 6LoWPAN stack. The MAC and PHY layers were specified by the IEEE 802.15.4 standard. A 6LoWPAN adaptation layer was built on top of that to provide the IPv6 network layer connectivity. The layered architecture allowed different transport and application layer protocols to be used on top of the IPv6. Palattella et al. [64] proposed a similar protocol stack and implemented an application layer using the UDP based Constrained Application Protocol (CoAP).

The ZigBee stack [88] also builds on the IEEE 802.15.4 PHY and MAC layers. However, not all IoT protocol stacks are built on top of the IEEE 802.15.4 standard. A Bluetooth stack [11] is based on the IEEE 802.15.1 standard and has been used in plenty of IoT applications. Z-Wave defines its own sub 1 GHz protocol stack for the IoT [85]. Also regular 802.11 (Wi-Fi) protocols can be found with IoT applications, even though they are quite resource hungry. It is certainly possible to implement IoT devices without IP connectivity. In fact, IEEE 802.15.4 can be used alone, without higher level protocols built on top of it. It may be the exact right choice for some IoT scenarios, where resource limited end devices are not able to implement higher level layers.

Protocols, such as IEEE 802.15.4, ZigBee, Bluetooth and 6LoWPAN, are optimised for short-range communications within sensor networks, and between IoT end devices and gateways. The name southbound connection is used to designate the connection from a gateway to end devices. However, there are other options for the link between a gateway and the core network. The connection from the gateway to the core network is called a northbound connection. The requirements for the northbound communication technologies largely depend on the environment. Some gateways may be located in places where wired media is the smartest way to enable connectivity. Other places may be so remote that connectivity is enabled using cellular or other long-ranged wireless technologies, such as Long-Term Evolution (LTE) or satellite communications. Northbound technologies could be a topic for another thesis, and therefore, are beyond the scope of this one.

3.2.1 IEEE 802.15.4

The IEEE 802.15.4 [38] standard specifies the MAC and PHY layers of the protocol stack. 802.15.4 radios are suited for wireless personal area networks (WPANs). They have low power consumption, low data rate and a communication range of around ten metres. One node in an 802.15.4 personal area network (PAN) operates as a PAN coordinator. There are two types of devices in the PAN: full functional devices (FFD) and reduced function devices (RFD). Only FFDs can operate as coordinators. RFDs are very simple devices that are intended to communicate with only one FDD at a time. The standard specifies two topologies: a star topology and a peer-to-peer topology. In the star topology, the coordinator establishes the network and other nodes communicate through it. In the peer-to-peer network, any node can communicate with any other node, as long as they are within communications range.

802.15.4 radios use a 16-bit or a unique 64-bit addressing. A 64-bit address is composed of a 24-bit organisational unique identifier (OUI) and a 40-bit portion assigned by the manufacturer. The MAC frame size is only 127 bytes long, and it is divided into MAC header, payload and frame check sequence fields. The MAC header may take a notable amount of the available frame size, especially if 64-bit addressing is used. The 16-bit addresses assigned by the PAN coordinator can alternatively be used. The MAC layer controls access to the physical radio channel. Its other features include acknowledgement and validation of frames. The MAC layer requests the PHY layer to perform a clear channel assessment (CCA) in order to make decisions on whether the channel is clear for transmission or not.

The PHY layer supports 27 channels. 16 channels are allocated in a globally used 2400–2483.5 MHz frequency band. Additionally, the frequency band of 902–928 MHz, with ten channels, is used in the USA and the 868–868.6 MHz frequency band with one channel, in Europe. In addition to transmission and reception of PHY protocol data units (PPDU), the features of the PHY layer include turning the radio on and off, channel selection, energy detection (ED), link quality detection and CCA. ED is used to see if there are other radios transmitting data on the channel. That information is also used as part of the CCA method. ED, signal-to-noise ratio or their combination is used to calculate the link quality indicator (LQI). According to the IEEE 802.15.4 specification [38], the LQI measurement shall be done on every received packet.

ZigBee

The ZigBee stack [88] relies on the IEEE 802.15.4 PHY and MAC layers. In addition, it defines a network (NWK) layer and an application (APL) layer. The NWK and APL layers communicate through an application support sublayer (APS). The network layer supports star, tree and mesh topologies. ZigBee defines three device types:

- ZigBee coordinator (IEEE 802.15.4 PAN coordinator)
- ZigBee router (IEEE 802.15.4 FDD)
- ZigBee end device (IEEE 802.15.4 FDD or RFD)

Every PAN has one coordinator, which also handles the network formation. Routers can forward data to other nodes in the network. Router and coordinator nodes are typically mains powered, as they cannot be put to sleep to save energy. End devices are the most constrained devices in the network. They are typically battery powered or rely on energy harvesting techniques. They also conserve energy by sleeping between duty cycles. ZigBee application areas include healthcare, home automation, industrial control and many others. The ZigBee specification defines the application profiles to provide interoperability between applications from different vendors [88]. The ZigBee Alliance has also defined an IPv6 based standard, which relies on a 6LoWPAN adaptation layer [89].

6LoWPAN

IPv6 over Low-power Wireless Personal Area Network (6LoWPAN) [71] is a set of specifications aiming to embed IPv6 connectivity into sensor networks and M2M devices. The 6LoWPAN utilises the IEEE 802.15.4 standard on the lower layers of the stack. The maximum transmission units (MTUs) in IPv6 are 1280 bytes, whereas IEEE 802.15.4 MAC frames have only 127 bytes, from which the header may consume a notable amount. For that reason, 6LoWPAN defines an adaptation layer on top of the 802.15.4 MAC layer to perform fragmentation of the large IPv6 packets. The name 6LoWPAN comes from a now concluded IETF working group [40]. However, the work on IPv6 connectivity over constrained networks has not stopped.

There is currently another IETF working group developing drafts and RFC's for similar topics [39].

3.2.2 IEEE 802.11

IEEE 802.11 [37] is a family of standards for wireless area networks (WLANs). It was designed for laptops and other mobile devices requiring high-speed wireless Internet connectivity. Like IEEE 802.15.4, 802.11 also specifies the PHY and MAC layers of the protocol stack. The frequency bands vary depending on the version of the standard. The most common frequency band being 2.4 GHz, but some versions of the standard operate on the 5 GHz band. The 2.4 GHz frequency band overlaps with the IEEE 802.15.4 channels, except for channels 25–26. Interference on other channels might be problematic for 802.15.4 PANs, because the transmitting power of the IEEE 802.11 devices is much higher [73]. The Wi-Fi Alliance [80] promotes and certifies 802.11 WLAN standards for commercial use. That is also the reason why IEEE 802.11 devices are often referred to as Wi-Fi devices. In this thesis, both WLAN and Wi-Fi are used to denote the IEEE 802.11 standards.

An IEEE task group [36] is working on an 802.11ah specification, which is an amendment to the 802.11 standard. 802.11ah is particularly interesting for the IoT and M2M applications, because of its frequency band, communications range and low energy consumption. 802.11ah avoids overlapping with many existing IoT frequency bands, as it is intended to operate on the 900 MHz band. Its throughput is less than with the traditional WLANs, but it has a communications range of nearly twice as long. Sub 1000 MHz frequencies also have more robust signal propagation, making them ideal for IoT and M2M applications. Aust et al. [9] considered sensor networks, backhaul networks, cellular off-loading, M2M communications and rural communications as potential use cases for the 802.11ah.

3.2.3 Bluetooth

Bluetooth offers an alternative to the IEEE 802.15.4 based protocol stacks. Especially the Bluetooth Low Energy (BLE), introduced in Bluetooth core specification version 4.0+ [11], is suitable for IoT applications. "Classic" Bluetooth is referred to as basic rate (BR) Bluetooth in the specification to distinguish it from the low energy (LE) version. Sometimes, the notation BR/EDR is also used, where EDR denotes enhanced data rate. Some devices may operate in dual mode, meaning that they can communicate with both the LE and BR versions of Bluetooth. Bluetooth devices broadcast advertisement messages periodically. The advertisement period can be set between 20 ms and 10.24 s in 0.625 ms intervals. The advertisement messages are used to form connections between devices, and as the name indicates, they can also be used to send advertisements. IoT applications can utilise advertisement messages to send periodic sensor data broadcasts. Currently, Bluetooth is supported by most of the mobile phones and laptops, and due to the small energy footprint of the BLE, it is becoming more popular in tiny sensor nodes, as well. There is also work in progress to bring IPv6 connectivity to the BLE [57].

Bluetooth BR/EDR

According to the Bluetooth core specification [11], Bluetooth BR devices are either masters or slaves. One master device can have up to seven slave devices in a so-called piconet. Two or more piconets can form a larger network called scatternet. In a scatternet, a device that operates as a master in one piconet, operates as a slave in the other piconet(s). Bluetooth operates in the 2.4 – 2.483.5 GHz frequency band. Bluetooth uses frequency-hopping spread spectrum (FHSS) on the available 79 channels. The hopping rate during transmission can be as high as 1600 hops per second. The adaptive frequency hopping (AFH) method avoids interference by performing hopping on less crowded channels. The communication range of a Bluetooth device is 10–100 meters. BR devices have data rates up to 1 Mb/s. If EDR is enabled, then the data rates are 2–3 Mb/s. Bluetooth applications communicate through application profiles, which include headsets, health devices, file transfers and printers, to name a few. For example, a PC needs a headset application profile in order to properly connect and use a Bluetooth headset.

The two mandatory protocols are the Link Management Protocol (LMP) and the Logical Link Control and Adaptation Protocol (L2CAP). The LMP sets up and controls the radio link between two devices. The L2CAP defines several transmission modes for the packets. The basic mode supports payloads as large as 64 kB with the default MTU being 672 bytes. Other modes are retransmission, flow control, enhanced retransmission and streaming mode (SM). SM offers unreliable communication and thus, does not support retransmission or flow control. [11]

Bluetooth Low Energy

Bluetooth Low Energy (BLE) [11] was designed for devices that require very low power for their operation. These include, but are not limited to, devices, which are powered by coin cells or energy harvesting techniques. Now, more and more smart phones have started using BLE, as well. BLE operates on the same frequency band as Bluetooth BR, but frequency hopping uses only 40 channels. The data rate is 1 Mb/s (the same as with BR). The core specification (4.0+) [11] specifies the Security Manager Protocol (SMP) and Generic Attribute Profiles (GATT) for the BLE. The SMP generates and stores the identity and encryption keys for communicating on an L2CAP channel. In BR, the link manager has similar functionality. The functionalities of GATT include communicating with peer devices and discovering services. It also has similar observer behaviour to that introduced in Section 2.2.2, where CoAP Observer protocol was covered. GATT specifies two types of observe messages: notifications and indications. When using notifications, a BLE server, which can be, e.g. a temperature sensor, sends messages repeatedly to a BLE client. Indications behave similarly, except they require acknowledgement from the client device.

3.3 Future connectivity

Long-range cellular IoT technologies bring an alternative to previously mentioned short-ranged communication technologies. Even though the two technologies are competing, they can complement each other. They offer alternative ways of accessing the IoT just like there are alternatives for accessing the traditional Internet using cellular Internet technologies, home WLAN gateways and so on.

Cellular IoT devices form Low-Power Wide-Area (LPWA) networks. LPWA systems require a long battery life, low device cost, low deployment cost, full coverage and support for massive numbers of devices. LPWA devices communicate directly with a base station and do not need a gateway device like the aforementioned short-range technologies. However, base stations may offer similar services as gateways do. Current LTE technologies are not suitable for the low power end devices, and even the more powerful gateways struggle, when there are many simultaneous devices sending data through the gateway's LTE interface [18].

There are several LPWA standards in progress. They are being developed by keeping the constrained nature of IoT devices in mind. Examples include, LTE-M and EC-GSM, which are expected to be available in 2016. LTE-M uses a bandwidth of 1.4 MHz, with a data rate of less than 1 Mbps. It offers a communications range up to 11 km. The narrowband version of the LTE-M supports a 200 kHz bandwidth with a communications range up to 15 km and a data rate up to 150 kbps. EC-GSM has a lower data rate of up to 10 kbps, using 2.4 MHz bandwidth. In the future, 5G is also expected to offer connectivity for cellular IoT applications. [58]

As the 2.4 GHz frequency band suffers from overcrowdedness, some of the IoT PHY layer protocols are moving to the Sub-1 GHz range [9]. The IEEE 802.15.4 specification [37] already supports the 868.0–868.6 MHz and 902–928 frequency bands. One interesting upcoming Sub-1 GHz standard is the IEEE 802.11ah [36] that was presented in Section 3.2.2. It is interesting, in a sense, that today, many homes and offices already have Wi-Fi access points for people to access the Internet. Perhaps in the future, Things can access the Internet using the same access points that people use, but on different frequency bands.

4 IoT gateway

The main task of an IoT gateway is to connect "Things" to the Internet. The multitude of different radio technologies demand gateways that can support multiple radio interfaces and communication protocols. The PHY and MAC layer protocols, that were introduced in Chapter 2, are most likely the ones that need support from the gateway side. Still, it might not be enough, because the sensor nodes might use other protocols on top of the PHY and MAC layers. In those cases, the gateways might need to understand the higher level protocols, as well. The idea of a gateway is not a new one as similar devices were already used in the early Internet [15]. Also WSNs have been utilising gateways for a long time. In the WSN literature, gateways are referred to as sinks [2].

Different kinds of IoT gateways have been presented in academic papers in the past years. Zhu & al. [87] developed an IoT gateway for a ZigBee based WSN. The northbound interface to the Internet was actualised using a GPRS module. Datta et al. [19] used multiple southbound and northbound interfaces to communicate Sensor Markup Language (SenML) data in a RESTful system. There are also many companies involved with the development of IoT gateways, already. For example, Digi International [22] offers various IoT gateways, which provide connectivity and management to their own ZigBee based XBee product family. Intel has variety of gateways for different target markets such as industry, transportation, energy and hobbyists [41].

There is, however, a big problem with the aforementioned gateways. They are usually designed for a very specific purpose, i.e. they support only one or a small set of protocols and radio interfaces. There are, however, a multitude of sensors using different protocols and this will quickly lead to a situation where one needs a new gateway for every new use case. The ideal situation is that one can take an IoT device out of one environment and place it into another, while keeping similar connectivity and functionality. Zachariah et al. [86] aimed to accomplish this by designing a mobile gateway. The idea is that normal smart phones would function as gateways, i.e. anyone using a smart phone would also offer gateway services for sensors and other devices. They used Bluetooth Low Energy (BLE) as the communication technology between the gateway and the sensors.

According to Chase [16], it is unlikely that there will only be one IoT standard standing, in the end. There are plenty of them now and there will be plenty of them in the future. It also justifies the use of an IoT gateway as it is a way to support multiple IoT standards and protocols.

4.1 Gateways and routers

Cerf and Kirstein noted in [15] that the IoT faces similar gateway challenges that the early Internet encountered, in the 1970s. Back then, it was not certain whether gateways should be application-level gateways or just natively relay messages using some network layer protocols. It turned out that the gateways started adapting IP-based network protocols and evolved into the IPv4/IPv6 routers of the Internet that we know today. Now IoT needs to find its way to extend the existing Internet. The question is again, should IoT use network-level routers or application-level gateways?

In the former scenario the network-level support is directly implemented in the IoT devices. If the end devices were able to implement a network stack that includes IP-connectivity, then gateways did not have to process the messages in the application layer. Gateways would then perform as IoT routers and relay the messages like traditional routers. The previously discussed 6LoWPAN specification intends to bring IPv6 connectivity to low-power end devices. Gateways in 6LoWPAN are called border or edge routers. Border routers implement an IPv6 adaptation layer on top of the MAC layer. The adaptation layer slices large IPv6 packets into suitable sizes so that they can be carried in IEEE 802.15.4 frames. Additionally, border routers support neighbour discovery, link-layer & network routing, IPv6/IPv4 interconnection, firewall & access control and management & proxy services [71]. The use of IPv6 based end devices would enable similar routing functionality as is used in the Internet today. For example, the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [82] has gained some ground in the IoT/WSN together with 6LoWPAN.

In the latter scenario, the IoT end devices would communicate with gateways using non-IP protocols, such as IEEE 802.15.4, ZigBee or Bluetooth. Application-level gateways would then process the messages and pass them on using network level protocols, e.g. IPv6. Furthermore, the payload can be wrapped into even higher level protocols such as HTTP or CoAP. More about this approach is discussed in Chapter 5, where IoT gateway design is presented. These two approaches are also discussed a bit more, in the conclusion, in Chapter 6.

4.2 Gateway functionalities

This section introduces some important functionalities of an IoT gateway. It is not an exhaustive list of all the functionalities, as they also largely depend on specific applications. Furthermore, this is certainly not a definitive list of functionalities that an IoT gateway must have. In the end, it is up to the system architects and engineers to decide which functionalities should be the responsibilities of an IoT gateway, as some of the functionalities may also be implemented in the end device or in the back end.

4.2.1 Edge computing

Edge computing, also known as fog computing, is an extension of the cloud computing paradigm. Computation tasks that are normally executed in the cloud are brought to the network edge. It reduces latencies, as computation is done closer to the user. Edge computing is especially suitable for devices, which are mains powered. Sensor nodes are typically incapable of executing demanding computational tasks. Gateway devices, on the other hand, are often mains powered and equipped with more processing and storage capabilities for edge computing purposes [13].

4.2.2 Dataflow control

IoT gateways have one or more southbound and northbound interfaces. Southbound interfaces face the IoT end devices and northbound interfaces connect to the cloud services. For example, an IoT gateway may offer IEEE 802.15.4, ZigBee and Bluetooth as southbound interfaces and LTE and Ethernet as northbound interfaces. The IoT Gateway may also support multiple higher level protocols. There may be a need to send data from the IoT devices to multiple destinations using, for example, both CoAP and HTTP.

IoT gateways can also be thought of as being part of a software defined networking (SDN) system. SDN decouples the control plane and the data plane in the network. SDN uses centralised intelligent devices, that control how the underlying devices relay data. In the IoT scenario, the gateways take the roles of controllers, which control the behaviour and dataflows of the underlying network of Things. Stojmenovic [74] applied the SDN idea to a fog computing architecture, where one device in the fog took the role of the controller.

In cases, where the gateway relays information between sensors and sensor networks, it is important that the gateways support data aggregation. The goal of data aggregation is to save energy in the receiving ends by filtering and combining the data before sending.

4.2.3 Proxy functionality and caching

Proxies are often implemented in gateways. They offer essential functionality for the low-power and lossy networks (LLNs). Proxies are relevant if the clients cannot or should not access the physical devices directly. The use of a proxy may reduce response time, network bandwidth and energy consumption of an end device. There are a couple of benefits in using proxies. Firstly, they can be set to listen for notifications from a node. In this case, the nodes can decide when to send data and when to save energy. Secondly, proxies can serve multiple clients, which allows the node to only communicate with the proxy. This saves the nodes' energy, memory and processing resources, because there will only be one client (proxy) accessing it. Sometimes protocol conversions are needed as devices prefer energy-efficient protocols, whereas human users may prefer the user-friendly ones. For example, Mainetti et al. [53] used a proxy server to convert CoAP notifications into WebSocket notifications that could be read from a Web browser. CoAP-to-HTTP or HTTP-to-CoAP are other common protocol conversions.

Proxies can also support caching. It is another powerful method for saving energy in the node. Notifications or events coming from the node can be stored for a certain period of time. If clients want to request a certain resource, the proxy can serve the resource from its cache instead of passing the request to the node. An important thing is to keep the cached values up-to-date and that is the responsibility of the proxy. CoAP supports all of these functionalities natively.

CoAP [72] defines two types of proxies: a forward-proxy and a reverse-proxy. Forward-proxies perform requests on behalf of clients. A forward-proxy is selected by a client by using a Proxy-Uri option. The forward-proxy consumes the Proxy-Uri option and creates the Uri-Host, Uri-Port, Uri-Path and Uri-Query options, which are forwarded to the actual server. Reverse-proxies stand in for the actual server and a client may not be aware that it is communicating with a reverse-proxy instead of the actual server. Proxies can simply perform CoAP-to-CoAP mapping or even do cross-protocol translations. As an example of a cross-protocol translation, a proxy may take the client's request in HTTP, translate it into a CoAP request and forward it to the actual server. The actual server will then send a CoAP response, which will be translated into an HTTP response by the proxy, and then finally gets forwarded to the client. Caching is also supported by CoAP. CoAP uses a freshness model to arrange caching. The freshness of a message is determined by the expiration time defined by the Max-Age option. If the response is fresh and it can satisfy the request, then it is sent to the client instead of contacting the actual CoAP server. If the proxy has disabled caching, then the requests will be passed directly to the actual server. The Max-Age option can also be set to zero, which avoids caching that particular message.

4.2.4 Resource discovery

Resource discovery is an important functionality in M2M and IoT applications as there are multitude of different types of sensors and actuators offering their resources. Resource discovery can be done in different layers of the network depending on the network size and type. The essential thing is that the devices should somehow be able to expose their resources, so that a client using an IoT application will be able to know about them.

CoAP [72] supports resource discovery and it follows the CoRE link format as specified in [70]. The link format specifies a URI prefix "/.well-known/core" for listing all the resources that the server is offering. Resource directories can be used to gather resource descriptors from multiple servers in the network. Resource directories are hosted in more capable devices such as gateways, which can poll the servers or receive notifications about the resources. The link format also supports filtering the resources by using the query section in URIs as seen in Section 2.2.2. It is useful when the client is only interested in specific types of resources. In [53], Mainetti et al. built a Web of Things architecture where device discovery was implemented in a CoAP gateway proxy.

Datta et al. [20] view CoAP as lacking some important discovery functions. Firstly, the CoAP server has the ability to expose its resources, but there is no mechanism to announce its own existence. The clients need to have previous knowledge that such a server exists. Secondly, it is not specified how a client can remotely look up the resource directory. Thirdly, there is a scalability issue with the resource directory and CoAP. For example, in larger networks the gateways should be able to propagate their resources to higher level resource directories. Due to the shortcomings of built-in resource discovery of CoAP and other protocols, Datta et. al [20] proposed a technology independent architecture for the resource discovery. In their architecture, the end nodes expose resources by sending notifications to a proxy layer. The clients use search engines to find resources that have been registered to the configuration registry by the proxy.

4.2.5 Security

As noted in [7], [30] and [55], security and privacy are open issues in the IoT research. IoT gateways are attractive targets for various attacks as they connect all sorts of constrained devices to the Internet. In [81], secure booting, access control, device authentication, firewalling, and updating and patching were listed as mechanisms to secure IoT devices. Frantti et al. [29] made a risk analysis of different security threats facing the IoT and defined security requirements for an IoT edge router. It is essential to understand that security is not an add-on, but a fundamental part of the underlying architecture [29], [55]. Security issues are not in the scope of this thesis, but it is important to take them into account, when designing IoT systems.

4.3 Gateway management

Like IoT end devices, gateways also need to be managed. There might be security fixes, new features, device settings or new dataflow profiles that need to be remotely changed or updated in the gateway. It is also important to be able to monitor the status of the gateways. This section introduces some of the well-known device management standards.

4.3.1 Management standards

The standards presented in this section are often associated with end devices. However, they are not strictly designed for specific types of devices and they are well suited for gateway management, as well.

TR-069

Broadband Forum's (previously DSL Forum) TR-069 technical specification [24] describes a communication protocol between customer-premises equipment (CPE) and an auto-configuration server (ACS). The protocol also specifies security and CPE-related management functionality. CPE WAN Management Protocol (CWMP) uses HTTP requests with GET, PUT and POST methods to handle device management tasks. Management requests are always initiated by the CPE device to which the ACS responds with management messages. CPE encloses a heterogeneous collection of different kinds of devices. TR-069 can be used with home gateways, set-top boxes, VoIP phones, game consoles and other kinds of electronic devices [66].

OMA device management

Open Mobile Alliance Device Management (OMA-DM) [63] specifies an architecture, requirements and a protocol for end device management. OMA-DM builds on RESTful architecture using HTTP as the application layer protocol. The requirements include mechanisms to handle management tasks, interoperability, administration and security. The management tasks include configuring device settings, managing and upgrading the device with software updates, retrieving device information for fault management and so on.

OMA Lightweight M2M

OMA Lightweight M2M (LWM2M) [61] is a management standard aimed for managing low-power embedded devices. As a comparison to the aforementioned standards, LWM2M extends the range of devices that can be managed. LWM2M devices are constrained in terms of network bandwidth, computing power, memory, battery and cost. Also, more complex devices, such as gateways and hubs, can be managed with LWM2M. Management tasks include switching the device on and off, software updates, configuration, maintenance, error recovery, monitoring and data queries [47].

LWM2M defines a client-server communication architecture. The architecture also includes bootstrapping and smart-card modules [61]. LWM2M defines a set of high-level functional requirements and a couple of security requirements. High-level functional requirements specify, for example, what kind of information is transferred to and from the device and which features should be managed. Security requirements include authentication, authorization, data integrity and confidentiality [62].

An example implementation was introduced in [47]. The implementation is based on RESTful architecture. CoAP is used as the application layer protocol. LWM2M client is composed of objects, which contain different resources. Resources can be read, written or modified by using CoAP GET, POST, PUT and DELETE methods. CoAP messages can be delivered using 6LoWPAN, Wi-Fi or a short message service (SMS). End-to-end security builds on the Datagram Transport Layer Security (DTLS), which provides a secure channel between a LWM2M client and a LWM2M server. Secure communication can be built on public key or pre-shared key technologies.

4.3.2 Management protocols

The previously presented TR-069 and OMA-DM use HTTP as the application layer protocol. The devices run RESTful services that can be accessed using the request methods that were mentioned in Section 2.2.1. Similar RESTful services can be implemented using CoAP, as was the case in [47]. However, CoAP has a broader range of target devices than HTTP, due to its suitability to low-power and lossy networks.

MQTT brings another interesting perspective to gateway management. Management or configuration messages can be delivered in publish-subscribe fashion using a centralised broker. In [46], Kim et al. proposed an MQTT based architecture for device management. They used an IoT gateway as a broker to forward control messages to different IoT devices. The same idea can also be applied to control gateways themselves. Unfortunately, there are currently no management standards that build on the MQTT protocol.

4.4 Virtualising the gateway functionality

Network Function Virtualisation (NFV) is a novel technique to reduce time to market of new services and to improve network flexibility and manageability. NFV decouples software and hardware by a virtualisation layer, which abstracts the physical resources. From a hardware perspective, there can be a very heterogeneous set of devices such as smartphones, gateways, routers and set-top boxes. Through a virtualisation layer they all can have the same functionality and behaviour with a consistent management interface [33].

Docker [23] is a good example of a virtualisation tool that can be used for gateway devices. Docker wraps all the needed functions and packages into a container. Containers can be run on any platform supporting Docker, without the need to do platform specific modifications to the container. Docker containers are much more lightweight than the traditional virtual machines. They can, therefore, support manageability and maintainability in small devices like IoT gateways. Containers are built using instructions contained in a Dockerfile. To provide new functionalities for a network device, new containers can be seamlessly deployed by using a centralised image repository. [23]

5 IoT gateway development and deployment

The first goal of the empirical part is to develop an IoT gateway that provides connectivity between end devices and back end servers. The second goal is to deploy eight gateways in a laboratory environment. The gateways will collect data from a heterogeneous set of end devices, in an area of approximately $640m^2$. The long-term goal is to have an IoT testbed that can be used to study and compare, for example, different communications protocols, power consumption and localisation algorithms. Figure 5.1 presents the gateways designed in this thesis. The gateway on the right is one of the eight gateways deployed for the laboratory demo. The one on the left is used as a standalone gateway for demonstrations and for introducing new gateway functionalities before they are deployed in the lab network. The network topology is presented in Section 5.3. It is important to note that the development of the gateway and the systems related to it have been developed in a project with other researchers and developers. There have been a lot of people contributing to different parts of the project, from the planning state, to the implementation and testing phases. The author of this thesis has focused on defining use cases, developing gateway software and deploying gateways in the laboratory environment. The software tasks included the design of sensor interfaces in the gateway, internal data processing and external gateway communication. Other components, such as visualisation and back end, have not been designed by the author. However, they are also presented in this chapter, as they are relevant for providing the big picture of the system.

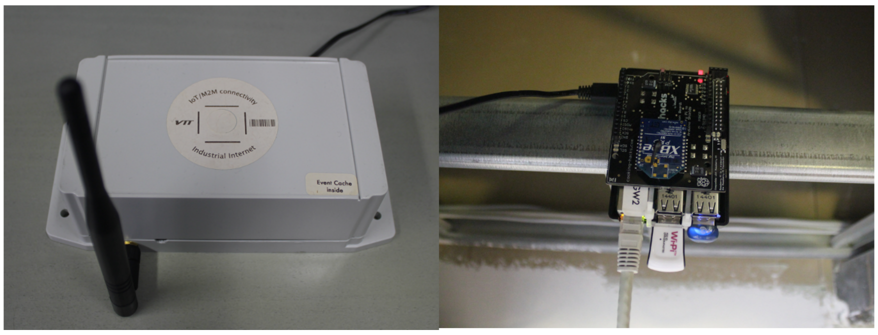


Figure 5.1: The designed gateway with and without covers.

5.1 Gateway architecture

The main focus in the gateway design is the software. The Distributed Decision Engine (DDE) was selected as the software architecture for the gateways. It will be discussed in detail in Section 5.1.1.

Raspberry Pi 2 was selected as the computing platform for the gateway as it is a highly customisable board in terms of software components and hardware peripherals. Raspberry Pi 2 also has a significant boost in computing capabilities, compared to the previous versions, as it has a 900 MHz quad-core ARM Cortex-A7 CPU and an increased amount of memory. It supports Secure Digital memory cards up to 32 GB, so it can operate as a local storage for logs and cached data [67].

The gateway software has also been tested on Intel Galileo Board [42]. It enables the use of various IoT peripherals, as it is compatible with Arduino shields. Programs on Galileo can be written like any Arduino program using the Arduino Integrated Development Environment (IDE). The Arduino programs run on top of an embedded Linux operating system. Therefore, Intel Galileo supports embedded Linux software development, as well as quick prototyping with the Arduino IDE.

5.1.1 Distributed Decision Engine

The Distributed Decision Engine (DDE) [69], [52] forms the underlying architecture of the gateway. The DDE enables rapid development of new features and adds modularity to the gateway architecture. Figure 5.2 presents the general DDE architecture. It uses a publish-subscribe model, which was presented in Section 2.3. In DDE, information producers collect data from sensors and publish them as events through a module named Event Cache. The Event Cache distributes the events to all consumers who have subscribed to them. Consumers will then process the events or send them further to a remote server.

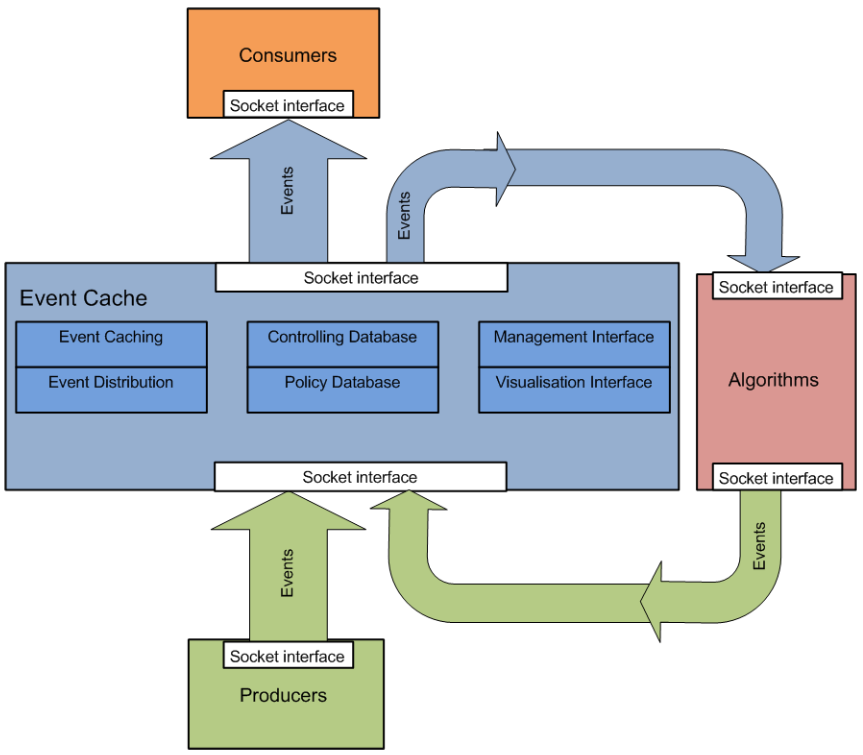


Figure 5.2: Distributed Decision Engine. Adapted from Rautio et al 2013 [69].

Event Cache is not just for caching and distributing information. The visualisation interface can be used to view the current status of different DDE modules. The management interface can be used to modify producer registrations, consumer subscriptions and policies. Access by different consumers can be controlled using policies that are stored in the Event Cache's policy database. The controlling database has information of all registrations and subscriptions made by producers and consumers.

The DDE also supports modules that work both as consumers and producers. In fact, the algorithm module, specified by the DDE, operates that way. It receives an event from the Event Cache from its consumer interface, does some processing to the data, and finally publishes the processed data, as a new event, through its producer interface [52].

5.1.2 Data format

The producers use JavaScript Object Notation (JSON) to communicate sensor data to the Event Cache. The Event Cache relays them onwards to consumers who have subscribed to that event. Consumers can further process the data contained in the JSON or send it further on without changes. The JSON packets are then appended to a CoAP, HTTP or MQTT request. The data format has the following structure:

```
{
    payload: {
        rssi: float
        tagID: string
        gwID: string
        timestamp: string
        samples: {
            sensor1: integer
            sensor2: integer
            ...
            sensorN: integer
        }
    }
}
```

XBee, Wi-Fi and Bluetooth based producers use the above format, although not all end devices have sensors. In that case, the samples section is empty. The format is very lightweight and samples include only the name and the value of the sensor. Currently, the sensor values represent the analog-to-digital conversion (ADC) values, which need to be converted to the needed unit in the application side. As noted in Section 2.3, the producers should not change the data format arbitrarily, as it may break the subscribing modules.

5.1.3 Hardware interfaces

Hardware interfaces allow different peripherals to be connected to Raspberry Pi. Raspberry Pi 2 has four USB ports, which can be used to connect, e.g. Wi-Fi dongles and other USB devices. 40 GPIO pins enable custom made or off-the-shelf hardware components to be interfaced with the Raspberry. GPIO pins include UART, i2c, SPI and i2s audio interfaces. Peripherals can use 3.3 V or 5 V pins as their voltage supply.

USB ports and the GPIO UART are needed for the selected peripherals. Wi-Fi and Bluetooth dongles are connected to USB ports and the XBee module is connected to the UART, using a specialised shield [17].

One thing to consider, when adding peripherals, is the current consumption. Current consumption has to be within the limits of the power supply's output current, otherwise the gateway will not work as expected. The output current for the Raspberry Pi power supplies is usually around 1-2 A.

5.1.4 Operating system

There are plenty of options for an operating system for Raspberry Pi 2 due to its increased computing capabilities. Raspbian is a popular operating system for Raspberry Pi and it is also officially supported by the Raspberry Pi foundation. Third party operating systems include, for example, Ubuntu Mate and Windows 10 IoT core [68]. Furthermore, hobbyists often use Archlinux, as it is a lightweight operating system that can be customised for a specific purpose [5]. Raspbian was selected for this work, as it offered many important tools by default, and it was a quick way to start the software development phase.

5.2 Interface design

The actual gateway instantiation of the DDE does not require that all of the specified components of Fig. 5.2 are implemented. Connectivity can be achieved by implementing only the needed producers and consumers around the Event Cache. The end devices use IEEE 802.15.4, IEEE 802.11n and Bluetooth Low Energy (BLE) radio technologies. Therefore, a producer module is created for each one of them, in the gateway. The task of a producer is to receive and parse a frame that is coming from one of the aforementioned technologies. The producer then appends needed metadata, such as timestamps and RSSI values, to the JSON payload. Consumer modules are implemented for CoAP, HTTP and MQTT to enable connectivity with remote servers. Figure 5.3 shows the different modules running in the actual gateway.

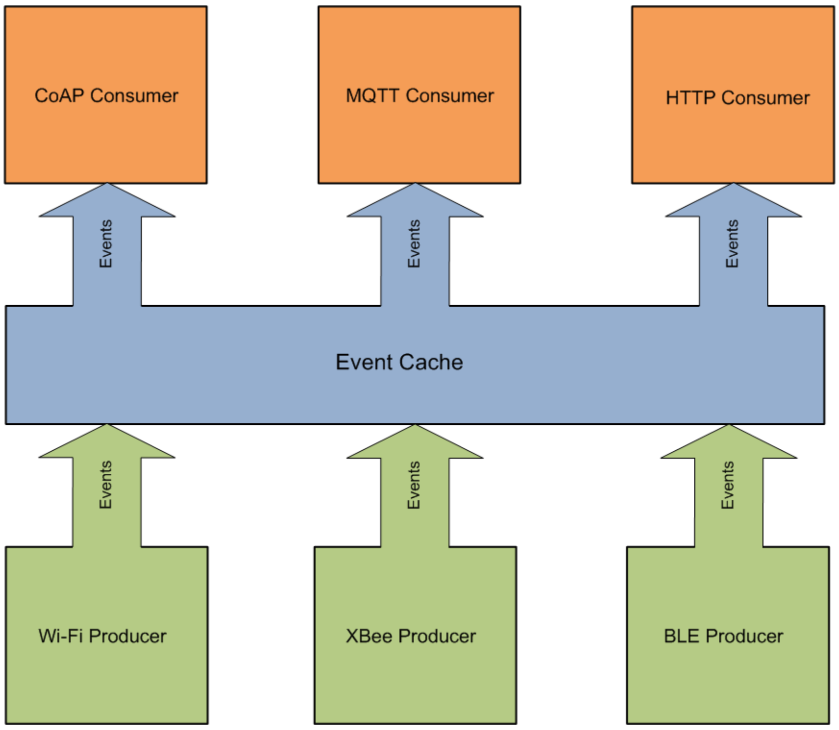


Figure 5.3: DDE modules inside the gateway

The producers have unique event identifiers that are used by the Event Cache to distribute events to correct consumers. Consumers subscribe to events during start-up. It is also possible to use one consumer to listen to multiple events. The modularity of the DDE allows manipulating dataflows by adding, changing and removing the event subscriptions. Python was used as the programming language, mainly because it makes prototyping much quicker than with the more efficient, lower level languages. The DDE provides the flexibility to use almost any programming language, as the modules communicate through socket interfaces.

5.2.1 Wi-Fi Producer

The task of the Wi-Fi Producer is to monitor traffic in the IEEE 802.11n frequency band. The pieces of information to be collected are the RSSI value and the MAC address of the end device. In order to perform meaningful monitoring on WLAN devices, we need to see all traffic on a given channel. A promiscuous mode lets us see all traffic from the network to which the monitoring device is attached. However, it is not enough to monitor only one network, as we want to see traffic from all devices including the ones that are not associated to any network. Moreover, we do not necessarily want to be attached to any access points while doing monitoring. It turns out that devices supporting the monitor mode can do exactly that. In the monitor mode, the wireless interface is able to listen to traffic from all the devices, in a specific channel. Unfortunately, experience has shown that not all wireless interface drivers support the monitor mode.

The interesting message types are probe requests, which are broadcasted by laptops and smart phones. Devices broadcast probe requests on all channels that are supported by the device. Therefore, we can set our monitoring device to listen to only one channel. Probe requests are used, for example, to gather information of the nearby access points. The frequency of sending probe requests varies greatly between different manufacturers. An important piece of information carried by the probe requests is the MAC address of the device. The MAC address is used together with the RSSI values to determine the location of the device. Linux supported Radiotap headers append additional information into the 802.11 frames including the important RSSI values.

The actual implementation of the Wi-Fi producer was coded in Python. Libpcap was used to capture incoming MAC frames including the Radiotap headers. A Pylibpcap python module made the C-based libpcap library callable from Python programs. That made it straightforward to parse the MAC frame and the Radiotap header in Python.

5.2.2 XBee Producer

The XBee Producer collects information coming from the XBee 802.15.4 nodes. The important pieces of information to be collected are sensor data, sender address and the measured RSSI value. Sensor values include, for example, temperature, humidity and brightness. The XBee modules are mounted on a shield that is placed on the Raspberry Pi. XBee 802.15.4 frames are read using a serial communication between the Raspberry Pi and the XBee module. The serial interface is shown as "/dev/ttyAMA0" in the Raspbian operating system. The interface is not hard to read, but there are existing libraries assisting reading characters and forming 802.15.4 frames. The pyserial library was used for initialising the serial communication and reading characters from the interface. In addition, an XBee Python library was used to help with parsing 802.15.4 frames. The RSSI measurement is conveniently supported by the XBee radio firmware, and the values are embedded into received 802.15.4 frames. There are also XBee libraries for several other programming languages including Java, C/C++ and Python.

5.2.3 BLE Producer

Bluetooth Low Energy (BLE) devices are set to broadcast sensor data periodically in advertisement messages. The task of the BLE Producer is to listen the advertisement messages. The implementation of the BLE Producer is very similar to the XBee Producer, with few exceptions. The BLE module is physically attached to a USB port of the Raspberry Pi and only temperature data is currently sent by the nodes. The incoming Bluetooth frames are read using a BlueZ Python module, which is the official Bluetooth stack implementation in Linux. Finally, the temperature data and the device ID are parsed from the Bluetooth frame.

5.2.4 CoAP Consumer

The CoAP Consumer operates as a CoAP client. It listens to events coming from the Event Cache and forwards them to a remote CoAP server. The CoAP Consumer is set to listen to events coming from all of the producers. In the case of WLAN data, the CoAP consumer sends a payload containing an RSSI value and a MAC address to the remote server using the PUT method. If an entry for the given MAC address already exists, then the server will update the RSSI value associated with the MAC address, in the database. Otherwise, the server will send a 4.04 (Not found) error. In that case, the CoAP Consumer will build a new POST request to create a new entry in the database. Now, all of the subsequent requests with that MAC address can be sent using the PUT method. However, this applies only to data coming from the Wi-Fi Producer. When dealing with unknown non-Wi-Fi devices, such as sensor nodes, the CoAP Consumer will not add a new device in the database. Libcoap was used to build a CoAP client, which was then utilised by the CoAP Consumer. Libcoap is written in the C programming language and, therefore, produces an external executable, which needs to be called from inside the CoAP consumer, for every request.

5.2.5 HTTP Consumer

The HTTP Consumer actually has the same functionality as the CoAP Consumer. It just provides an alternative protocol for transmitting data to a remote server. In some cases, it might even be preferable to use HTTP as there are plenty of options for Web servers available. CoAP is preferred in cases where the server has constraints in processing, memory or energy, or when the data is sent over low-power and lossy networks. The HTTP Consumer provides support for Transport Layer Security (TLS) and, therefore, is a good option when sending data over insecure networks. HTTP Consumer uses the Python Requests module for handling the HTTP connections.

5.2.6 MQTT Consumer

MQTT is used for real-time monitoring of sensor data. MQTT Consumer publishes sensor data to a remote broker, which forwards them to visualisation clients. This kind of setup allows for a quick way to implement new visual front-ends for, e.g. demos. MQTT Consumer subscribes both XBee and BLE events because they both contain sensor data that are needed in the current visualisations. WLAN events are also subscribed and aggregated in order to plot the amount of WLAN devices. The MQTT Consumer is implemented using the Python Paho MQTT library. The MQTT Consumer publishes data to following topics:

- /IoT_Demo/BLE/<tagid>
- /IoT_Demo/XBee/<tagid>
- /IoT_Demo/WLAN/<gwid>

The <tagid> placeholder is specific to the device from where the data is originating. The tagid of a BLE device is the hardware address (bdaddr) of the module. Accordingly, the XBee tagid is a 64-bit address from the XBee module. The <gwid> is the MAC address of the gateway, which forwards the Wi-Fi data. The data is aggregated on the application side and shown as the number of wireless devices that are within the communications range of a gateway.

5.3 Case examples

Eight gateways were connected to the laboratory network using an Ethernet connection. Figure 5.4 shows the network topology of the system. Gateways use IPv6 addressing, because it is considered as one of the enablers of the IoT as IPv4 addresses are running out [7]. The server can handle requests, which use CoAP, HTTP or MQTT protocols.

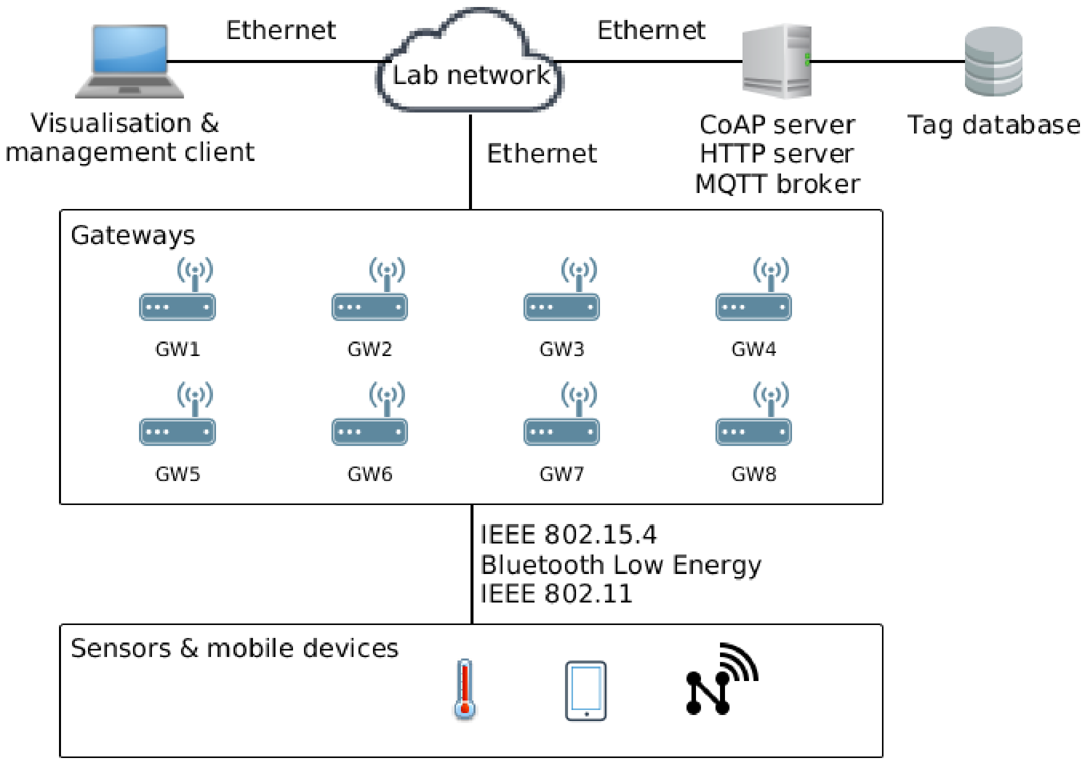
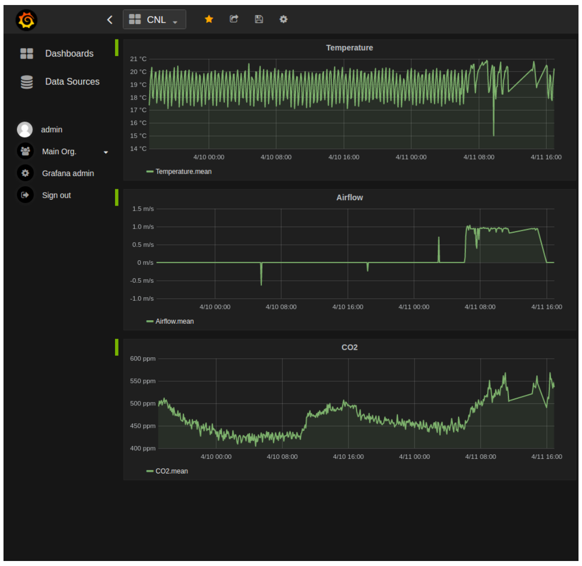
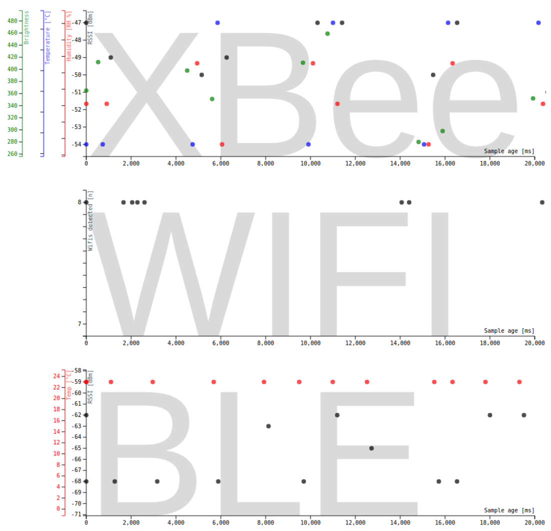


Figure 5.4: Network topology presenting end devices, gateways and the back end.

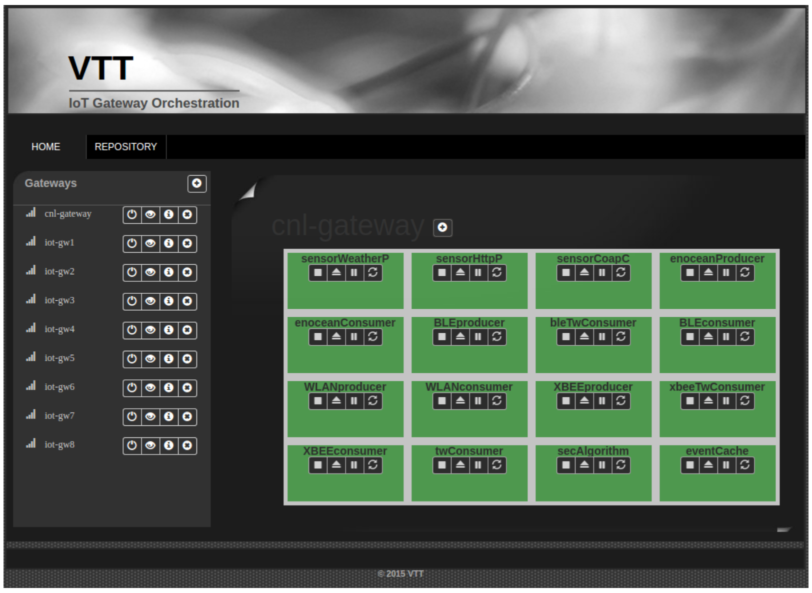
The tag database is used to store data sent by the sensors and mobile devices. The real-time monitoring follows the publish–subscribe paradigm discussed in Section 2.3. The monitoring client binds to an MQTT broker and subscribes to events coming from the MQTT consumer. The monitoring client receives notifications and the values are then updated on a graph, in real-time. The visualisation client can also show historical data to the user. Fig. 5.5a shows historical sensor data as graphs and Fig. 5.5b represents the real-time dataflow. Users can make HTTP requests from the visualisation client to the tag database for graphing data, from specific sensors. HTTP-to-CoAP mapping is implemented using proxy functionality similar to that, which was discussed in Section 4.2.3.



(a) Environmental monitoring



(b) Dataflow visualisation



(c) IoT gateway management

Figure 5.5: Screenshots of the user interfaces

Figure 5.5c is a screenshot from the IoT gateway orchestration tool. It is used to manage all of the different modules running inside the gateway, including the previously presented DDE modules. The management interface is implemented using Docker containers, which were discussed in Section 4.4. The basic functionalities include starting and stopping a container. It can also be used to deploy, update or remove a container. New containers are loaded from a remote image repository.

5.3.1 Case 1 - Environmental monitoring

There is a set of sensors in the lab monitoring several different properties. Some of the sensors are depicted in Figure 5.6.

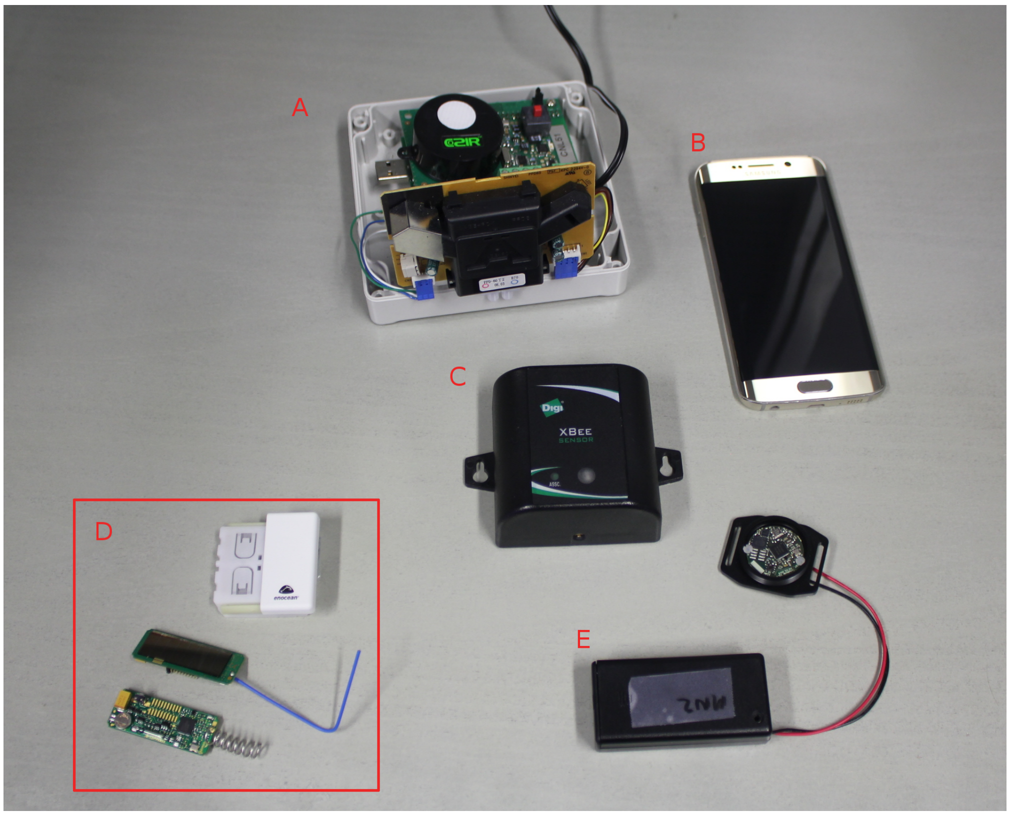


Figure 5.6: Some of the sensors used for environmental monitoring.

The most basic ones are temperature sensors, whereas more complex sensors include air flow (not pictured) and carbon dioxide (CO2) sensors. In the figure, the sensor marked with ”A” is a CO2 sensor. ”B” in the figure represents smart phones. They are not currently used for environmental monitoring and they have greater relevance in Cases 2 and 3. ”C” is an XBee sensor box, which contains temperature, humidity and brightness sensors. EnOcean sensors are grouped under the letter ”D”. They provide energy harvesting mechanisms for sensor data transmission. The DDE support for EnOcean sensors was developed by other members of the project, and therefore, are not covered in the previous sections. Finally, the sensor marked with ”D” is VTT’s TinyNode, which provides temperature readings using a Bluetooth Low Energy radio. Figure 5.5a depicts environmental monitoring data coming from the sensors. The sensor data is carried using Bluetooth and IEEE 802.15.4 based radios. The gateway interfaces that were presented in Section 5.2 play a key role in delivering data from the sensors to the back end servers and to the visualisation. The MQTT Consumer presented in Section 5.2.6 is used to provide data to visualisation clients in real-time. The visualisation works as a demonstration of the dataflow that goes through the gateway and can also be used to demonstrate the management functionalities of the gateway. The CoAP Consumer (or the HTTP Consumer) subscribes to sensor events and forwards them to a remote server for storage. Visualisation clients can then show historical data as graphs and plots.

5.3.2 Case 2 - Tracking an object/indoor localisation

Static sensors are simple in terms of localisation, as the location does not change over time. Mobile sensors and objects pose a challenge as they will move, thus, their location needs to be frequently changed in the database. There are scenarios where everyday items, such as keys or phones, need to be tracked and localised in indoor spaces. There are several different technologies for achieving indoor localisation. The most inexpensive way is to use the existing hardware and perform RSSI based localisation. The IoT gateways record RSSI values from all of the radio interfaces, hence allowing rough distance estimation and localisation on the server side. The location can be calculated when the broadcasting tag is heard by multiple gateways.

The weighted centroid localisation algorithm [12] was chosen for the case example. It is based on a Received Signal Strength Indicator (RSSI), which is measured in the gateway from incoming packets. All of the chosen technologies, namely Wi-Fi, BLE and XBee send periodic broadcasts that are captured by their respective producer modules, at the gateway. RSSI values are then sent to a remote server and the location is calculated using the weighted centroid localisation algorithm. The location is then updated in the tag database, and can be queried by the visualisation client. The prerequisite of the technique is that the locations of the gateways are known and stored in the database.

An earlier version of the gateway was used in a children's safety application, where their locations were approximately measured to ensure that they are in allowed areas during daycare [43]. The children used safety vests, which were supplemented by some processing, sensing and radio capabilities. The safety vests are still compatible with the gateway presented in this thesis.

There are alternatives to RSSI-based localisation. Some of the more advanced and accurate ones include time of flight (ToA) and time difference of arrival (TDOA) techniques [51]. Their distance estimation is based on the propagation time of the signal. They are, however, quite expensive to deploy, as the timing has to be very precise, and that requires specialised hardware. Although RSSI-based localisation is not a very accurate technique, it is an inexpensive option, as no extra hardware is required.

5.3.3 Case 3 - Crowd counting

Crowd counting is a technique used for counting how many people are in a given area. The examples include crowded areas such as shopping malls, rallies and festivals. State of the art crowd counting techniques use Wi-Fi signals to count the number of people in an area [21], [83]. People do not even need to have smart phones, as the systems are based on detecting changes that people cause on the Wi-Fi signals that are actively sent by the detectors. The aforementioned techniques are beyond the scope of this case example, and a more straightforward approach is used for rough crowd estimation. The crowd counting technique supported by the gateway counts the wireless devices that are sending probe requests. It, therefore, misses a lot of people, but can still provide information on which areas are more populated at specific times. The method counts all of the devices that are within the communications range of the gateway.

The Wi-Fi Producer is used for extracting information from probe requests. It is the same module as in the indoor localisation case. The difference is a new instance of the CoAP consumer, which forwards data to another application running on the server. Additionally, an MQTT Consumer is listening to the Wi-Fi events and forwards them to be aggregated and shown in the visualisation. The visualisation in Fig. 5.5b shows the amount of Wi-Fis heard by the gateway during short intervals. The downside of this case example is that some of the newer mobile devices keep changing their MAC addresses. It is a welcomed feature for security reasons, but makes crowd counting more challenging.

5.4 Discussion

The preliminary versions of the gateway were initially presented in [44], [45] and [43]. Since then, it has gone through many cycles of development and evolved into an easily manageable multi-purpose gateway. The publish-subscribe paradigm worked well as the message passing architecture. It was utilised in two places in the system, as the Distributed Decision Engine (DDE) handled internal communication in the gateway and the MQ Telemetry Transport (MQTT) protocol was used to bind gateways to visualisation clients, through a centralised broker. The DDE made it possible to easily control dataflows from the gateway to the outside world. In addition, modularity of the DDE enables quick development of new gateway functionality. The DDE is well suited to container-based management as DDE modules are easy to ship and deploy inside containers. The management interface allows quick loading and unloading of the functionalities. Eight gateways were successfully installed in the laboratory network, and they were ready to serve as a testbed for IoT research, as was the original plan.

CoAP support was built into the gateway, but was merely used between the gateway and the back end. The sensors, however, did not have CoAP capabilities, and therefore, the end-to-end communication was not purely CoAP. Instead, the end devices only used lower level protocols, such as IEEE 802.15.4, 802.11 and Bluetooth Low Energy to communicate with the gateway. The gateways processed the frames, read the payload content and appended it to a JSON data structure, along with the measured RSSI values and timestamps. The JSON data structures were then forwarded using CoAP, HTTP and MQTT protocols.

Gateways are essential in providing connectivity between sensors and the cloud. However, the gateways can do a lot more than that. In addition to receiving sensor data, the gateways monitored wireless LAN broadcasts. For every incoming packet, the gateways recorded the signal strength and timestamp that were required by the three case examples. The gateway provides ways for device management, caching, data filtering and aggregation, and edge computing. As of now, the user is expected to take actions based on the sensor data, but in the future, more responsibility could be given to actuator nodes. The gateway architecture supports creating and connecting actuator nodes as well as deploying intelligent algorithms for information processing. Therefore, some of the reasoning can already be done on the gateway side (edge computing), and more computationally demanding tasks can be passed to the cloud. Again, DDE is of great help in controlling where the data is processed.

The indoor localisation scenario in Case 2 did not require any additional hardware. The only piece of information needed was the RSSI value that was measured from every incoming frame by the gateway. Therefore, it was a very inexpensive way to provide approximate locations of objects. In terms of accuracy, there are better indoor localisation techniques available, such as systems based on ultra-wideband radios, which allow for taking accurate time-of-flight measurements. Similarly, the other two case examples represented useful scenarios, without the use of expensive hardware or overly complicated algorithms. Despite the simplicity of the crowd counting method, it can be used to monitor increased activity in specific areas. This can be combined with other data sources, such as CO2 sensors, to provide greater details about the activity in an area.

6 Conclusion

This thesis started from the big picture of the IoT by presenting existing architectural standards and the concept of the Web of Things. The application level communication paradigms and protocols that enable the Web of Things were also explained. Constrained Application Protocol (CoAP) was more carefully inspected as it is currently a trending topic on the IoT. The connectivity chapter narrowed the discussion down to Internet reference models and low-level communication protocols of the IoT. Important IoT specifications, such as IEEE 802.15.4 and Bluetooth Low Energy were covered. The discussion was further narrowed in the next chapter to cover topics related to the IoT gateway. Important features and functionalities of the IoT gateway were discussed, along with the existing device management standards and methods. The empirical part covered an IoT gateway design and implementation. Eight gateways were then successfully deployed and used in three case examples. Therefore, the initial goal to provide a platform for future IoT research was accomplished. The author of this thesis contributed to the use case definitions, gateway software development and gateway deployment. Important topics that were made by other project members, such as back end and visualisation, were briefly covered in order to provide a big picture of the system.

The questions that were raised in the introduction were: how to provide connectivity using an IoT gateway, what are the main functionalities of an IoT gateway, and how well does a gateway fit into the IoT architecture, in contrast to other options. The first question was answered by the theoretical and empirical parts of the thesis. Chapter 3 introduced the IoT protocol stacks, which were then used in the empirical part to provide end-to-end connectivity from a sensor node to the back end. The second question was partially answered by Chapter 4, which discussed IoT gateways and their functionalities. The functionalities also depend on the application domain, and therefore, differ between gateways. Also, not all of the functionalities introduced in Chapter 4 were needed in the gateway that was developed in the empirical part. A complete answer to the question would require a more extensive survey on different application domains and the identification of their requirements. The answer to the third question is not straightforward, either. Two different view-points emerged in the thesis for answering the question.

One approach is to compare the two architectures, where one architecture uses gateways and the another one uses direct communication between an end device and the access network, e.g. a base station. Cellular IoT is still in the research and development phase, but it shows intriguing future possibilities. However, the two architectures can co-exist, as was the case in the ETSI M2M high-level architecture introduced in Section 2.1. Similarly, people connect to the Internet using Wireless LAN standards and 4th generation mobile networks. Perhaps, a similar analogy works with the future IoT, as there are cellular IoT technologies in development, as well as current and upcoming short-range wireless standards. Therefore, it is likely that the technologies will supplement each other.

The second approach is to compare application-level gateways and network-level gateways, i.e. routers, as was done in Section 4.1. Network-level routing using IPv6 certainly shows significant potential in the IoT. However, it means that the end devices must implement a 6LoWPAN/IPv6 stack, which might not be possible for every device. Therefore, it leaves room for application-level gateways, as well. Furthermore, nothing rules out gateways that could handle both 6LoWPAN end devices and other non-IPv6 devices, in parallel.

The future looks bright for the IoT. There is active standardisation work going on to specify IoT architecture and protocols. The proposed cellular IoT technologies can expand the range of the future IoT applications, even further. Sub-1 GHz standards provide a less crowded channel for the IoT applications and there are new interesting specifications under development. Currently, it looks like the amount of new IoT specifications keeps on increasing, but there are also a set of well-established IoT protocols, such as IEEE 802.15.4 and CoAP, which have gained ground in many IoT applications. One future direction for gateway research is to further explore the possibilities of virtualisation, and how to use it, e.g. for device and network management. Although the IoT has been a hot topic for a long time, it is still in its early stages. Also, it is worth mentioning that it is not just a technology of the future, it is already here.

References

[1] Aghaei, S., Nematbakhsh, M. A., and Farsani, H. K. Evolution of the world wide web: From web 1.0 to web 4.0. *International Journal of Web & Semantic Technology 3*, 1 (2012), 1.

[2] Akyildiz, I., Su, W., Sankarasubramaniam, Y., and Cayirci, E. Wireless sensor networks: a survey. *Computer Networks 38*, 4 (2002), 393 – 422.

[3] Alahakoon, D., and Yu, X. Smart electricity meter data intelligence for future energy systems: A survey. *IEEE Transactions on Industrial Informatics 12*, 1 (Feb 2016), 425–436.

[4] Alam, M., Nielsen, R. H., and Prasad, N. R. The evolution of m2m into iot. In *Communications and Networking (BlackSeaCom), 2013 First International Black Sea Conference on* (2013), IEEE, pp. 112–115.

[5] Archlinux. Archlinux homepage. URL: `https://www.archlinux.org`, accessed: 6.4.2015.

[6] Ashton, K. That 'internet of things' thing. *RFiD Journal 22*, 7 (2009), 97–114.

[7] Atzori, L., Iera, A., and Morabito, G. The internet of things: A survey. *Computer Networks 54*, 15 (2010), 2787 – 2805.

[8] Atzori, L., Iera, A., Morabito, G., and Nitti, M. The social internet of things (siot) – when social networks meet the internet of things: Concept, architecture and network characterization. *Computer Networks 56*, 16 (2012), 3594 – 3608.

[9] Aust, S., Prasad, R. V., and Niemegeers, I. G. Ieee 802.11 ah: Advantages in standards and further challenges for sub 1 ghz wi-fi. In *Communications (ICC), 2012 IEEE International Conference on* (2012), IEEE, pp. 6885–6889.

[10] Banks, A., and Gupta, R. Mqtt version 3.1.1. *OASIS Standard* (2014).

[11] Bluetooth Standard 4.2. *Bluetooth Core Specification*, 2014.

[12] BLUMENTHAL, J., GROSSMANN, R., GOLATOWSKI, F., AND TIMMERMANN, D. Weighted centroid localization in zigbee-based sensor networks. In *Intelligent Signal Processing, 2007. WISP 2007. IEEE International Symposium on* (2007), IEEE, pp. 1–6.

[13] BONOMI, F., MILITO, R., ZHU, J., AND ADDEPALLI, S. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing* (2012), ACM, pp. 13–16.

[14] BOSWARTHICK, D., ELLOUMI, O., AND HERSENT, O. *M2M Communications: A Systems Approach*. John Wiley & Sons, West Sussex, United Kingdom, 2012.

[15] CERF, V. G., AND KIRSTEIN, P. T. Gateways for the internet of things: An old problem revisited. In *Global Communications Conference (GLOBECOM), 2013 IEEE* (2013), IEEE, pp. 2641–2647.

[16] CHASE, J. The evolution of the internet of things. *Texas Instruments Incorporated, White paper* (2013).

[17] COOKING HACKS. Raspberry Pi to Arduino shields connection bridge. URL: `https://www.cooking-hacks.com/raspberry-pi-to-arduino-shield-connection-bridge`, accessed: 6.4.2016.

[18] COSTANTINO, L., BUONACCORSI, N., CICCONETTI, C., AND MAMBRINI, R. Performance analysis of an lte gateway for the iot. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2012 IEEE International Symposium on a* (2012), IEEE, pp. 1–6.

[19] DATTA, S. K., BONNET, C., AND NIKAEIN, N. An iot gateway centric architecture to provide novel m2m services. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on* (2014), IEEE, pp. 514–519.

[20] DATTA, S. K., COSTA, R. P. F. D., AND BONNET, C. Resource discovery in internet of things: Current trends and future standardization aspects. In *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on* (Dec 2015), pp. 542–547.

[21] DEPATLA, S., MURALIDHARAN, A., AND MOSTOFI, Y. Occupancy estimation using only wifi power measurements. *IEEE Journal on Selected Areas in Communications 33*, 7 (July 2015), 1381–1393.

[22] DIGI INTERNATIONAL. Digi homepage, 2015. URL: `http://www.digi.com`, accessed: 4.9.2015.

[23] DOCKER. Docker homepage, 2016. URL: `https://www.docker.com`, accessed: 1.2.2016.

[24] DSL FORUM STANDARD TR-069. *CPE WAN Management Protocol*, 2004.

[25] ETSI SPECIFICATION 102 690 V2.1.1. *M2M functional architecture*, 2013.

[26] EUGSTER, P. T., FELBER, P. A., GUERRAOUI, R., AND KERMARREC, A.-M. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR) 35*, 2 (2003), 114–131.

[27] EVANS, D. The internet of everything. *Cisco IBSG, White paper* (2012).

[28] FIELDING, R., AND RESCHKE, J. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC-7230, June 2014.

[29] FRANTTI, T., HIETALAHTI, H., AND SAVOLA, R. Requirements of secure wsn-mcn edge router. In *Information Networking (ICOIN), 2013 International Conference on* (2013), IEEE, pp. 210–215.

[30] GUBBI, J., BUYYA, R., MARUSIC, S., AND PALANISWAMI, M. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems 29*, 7 (2013), 1645–1660.

[31] GUINARD, D., AND TRIFA, V. Towards the web of things: Web mashups for embedded devices. In *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences), Madrid, Spain* (2009), p. 15.

[32] HAHM, O., BACCELLI, E., PETERSEN, H., AND TSIFTES, N. Operating systems for low-end devices in the internet of things: a survey. *IEEE Internet of Things Journal PP*, 99 (2015), 1–1.

[33] HAN, B., GOPALAKRISHNAN, V., JI, L., AND LEE, S. Network function virtualization: Challenges and opportunities for innovations. *Communications Magazine, IEEE 53*, 2 (2015), 90–97.

[34] HARTKE, K. Observing Resources in the Constrained Application Protocol (CoAP). RFC-7641, September 2015.

[35] IEEE. P2413 working group. URL: `http://grouper.ieee.org/groups/2413/`, accessed: 23.2.2016.

[36] IEEE 802.11AH TASK GROUP. Status of Project IEEE 802.11ah, 2015. URL: `http://www.ieee802.org/11/Reports/tgah_update.htm`, accessed: 13.1.2016.

[37] IEEE STANDARD 802.11. *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 2012.

[38] IEEE STANDARD 802.15.4. *Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*, 2011.

[39] IETF. IPv6 over Networks of Resource-constrained Nodes (6lo), 2012. URL: `https://datatracker.ietf.org/wg/6lo/documents`, accessed: 8.4.2016.

[40] IETF. IPv6 over Low power WPAN (6lowpan), 2016. URL: `https://datatracker.ietf.org/wg/6lowpan/documents`, accessed: 8.4.2016.

[41] INTEL. Gateway solution brief, 2014. URL: `http://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/iot-gateway-solutions-brief.pdf`, accessed: 18.2.2016.

[42] INTEL. Intel galileo, 2014. URL: `http://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/iot-gateway-solutions-brief.pdf`, accessed: 18.2.2016.

[43] JUTILA, M., KARHULA, P., RIVAS, H., AND PANTSAR-SYVÄNIEMI, S. End-to-end safety solution for children enabled by a wearable sensor vest. *"Journal of Ubiquitous Systems & Pervasive Networks" 6*, 1 (2015), 33 – 39.

[44] JUTILA, M., RIVAS, H., KARHULA, P., AND PANTSAR-SYVÄNIEMI, S. Implementation of a wearable sensor vest for the safety and well-being of children. *Procedia Computer Science 32* (2014), 888–893.

[45] JUTILA, M., STRÖMMER, E., ERVASTI, M., HILLUKKALA, M., KARHULA, P., AND LAITAKARI, J. Safety services for children: a wearable sensor vest with wireless charging. *Personal and Ubiquitous Computing* (2015), 1–13.

[46] KIM, S. M., CHOI, H. S., AND RHEE, W. S. Iot home gateway for auto-configuration and management of mqtt devices. In *Wireless Sensors (ICWiSe), 2015 IEEE Conference on* (Aug 2015), pp. 12–17.

[47] KLAS, G., RODERMUND, F., SHELBY, Z., AKHOURI, S., AND HÖLLER, J. "Lightweight M2M": Enabling Device Management and Applications for the Internet of Things. *OMA LWM2M, White Paper* (2014).

[48] KOSTER, M., KERANEN, A., AND JIMENEZ, J. Publish-subscribe broker for the constrained application protocol (coap). Internet-Draft draft-koster-core-coap-pubsub-04, IETF Secretariat, November 2015. `http://www.ietf.org/internet-drafts/draft-koster-core-coap-pubsub-04.txt`.

[49] KOVATSCH, M. Demo abstract: human-coap interaction with copper. In *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on* (2011), IEEE, pp. 1–2.

[50] KUROSE, J. F., AND ROSS, W. K. *Computer Networks*. Pearson Education, Essex, England, 2013.

[51] LIU, H., DARABI, H., BANERJEE, P., AND LIU, J. Survey of wireless indoor positioning techniques and systems. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 37*, 6 (2007), 1067–1080.

[52] LUOTO, M., RAUTIO, T., OJANPERA, T., AND MAKELA, J. Distributed decision engine - an information management architecture for autonomic wireless networking. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on* (2015), IEEE, pp. 713–719.

[53] MAINETTI, L., MIGHALI, V., AND PATRONO, L. A software architecture enabling the web of things. *Internet of Things Journal, IEEE 2*, 6 (2015), 445–454.

[54] MAJANEN, M., KOSKELA, P., AND VALTA, M. Constrained application protocol profile for robust header compression framework. In *Energy 2015, The Fifth International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies* (2015), ThinkMind, pp. 47–53.

[55] MIORANDI, D., SICARI, S., DE PELLEGRINI, F., AND CHLAMTAC, I. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks 10*, 7 (2012), 1497–1516.

[56] MULLIGAN, G. The 6lowpan architecture. In *Proceedings of the 4th workshop on Embedded networked sensors* (2007), ACM, pp. 78–82.

[57] NIEMINEN, J., SAVOLAINEN, T., ISOMAKI, M., PATIL, B., SHELBY, Z., AND GOMEZ, C. Ipv6 over bluetooth(r) low energy. Internet-Draft draft-ietf-6lo-btle-17, IETF Secretariat, August 2015. `http://www.ietf.org/internet-drafts/draft-ietf-6lo-btle-17.txt`.

[58] NOKIA. LTE-M -Optimizing LTE for the Internet of Things, White Paper, 2015.

[59] ONEM2M. oneM2M Homepage, 2016. URL: `http://www.onem2m.org`, accessed: 2.2.2016.

[60] ONEM2M TS-0001-V1.6.1. *Functional Architecture*, 2015.

[61] OPEN MOBILE ALLIANCE. *Lightweight Machine to Machine Architecture*, 2013.

[62] OPEN MOBILE ALLIANCE. *Lightweight Machine to Machine Requirements*, 2013.

[63] OPEN MOBILE ALLIANCE STANDARD. *OMA Device Management Protocol v2.0*, 2016.

[64] PALATTELLA, M. R., ACCETTURA, N., VILAJOSANA, X., WATTEYNE, T., GRIECO, L. A., BOGGIA, G., AND DOHLER, M. Standardized protocol stack for the internet of (important) things. *Communications Surveys & Tutorials, IEEE 15*, 3 (2013), 1389–1406.

[65] PREECE, J., ROGERS, Y., SHARP, H., BENYON, D., HOLLAND, S., AND CAREY, T. *Human-computer interaction*. Addison-Wesley Longman Ltd., 1994.

[66] RACHIDI, H., AND KARMOUCH, A. A framework for self-configuring devices using tr-069. In *Multimedia Computing and Systems (ICMCS), 2011 International Conference on* (2011), IEEE, pp. 1–6.

[67] RASPBERRY PI FOUNDATION. Raspberry Pi 2 Model B. URL: `https://www.raspberrypi.org/products/raspberry-pi-2-model-b/`, accessed: 6.4.2016.

[68] RASPBERRY PI FOUNDATION. Raspberry Pi downloads. URL: `https://www.raspberrypi.org/downloads`, accessed: 6.4.2016.

[69] RAUTIO, T., LUOTO, M., MAKELA, J., AND MANNERSALO, P. Evaluation of autonomic load balancing in wireless multiaccess environment. In *Wireless Communications and Networking Conference (WCNC), 2013 IEEE* (2013), IEEE, pp. 1416–1421.

[70] SHELBY, Z. Constrained RESTful Environments (CoRE) Link Format. RFC-6690, August 2012.

[71] SHELBY, Z., AND BORMANN, C. *6LoWPAN: The wireless embedded Internet*. John Wiley & Sons, West Sussex, United Kingdom, 2009.

[72] SHELBY, Z., HARTKE, K., AND BORMANN, C. The Constrained Application Protocol (CoAP). RFC-7252, June 2014.

[73] SRINIVASAN, K., DUTTA, P., TAVAKOLI, A., AND LEVIS, P. An empirical study of low-power wireless. *ACM Transactions on Sensor Networks (TOSN) 6*, 2 (2010), 16.

[74] STOJMENOVIC, I. Fog computing: A cloud to the ground support for smart things and machine-to-machine networks. In *Telecommunication Networks and Applications Conference (ATNAC), 2014 Australasian* (2014), IEEE, pp. 117–122.

[75] SUDEVALAYAM, S., AND KULKARNI, P. Energy harvesting sensor nodes: Survey and implications. *Communications Surveys & Tutorials, IEEE 13*, 3 (2011), 443–461.

[76] TALEB, T., AND KUNZ, A. Machine type communications in 3gpp networks: potential, challenges, and solutions. *Communications Magazine, IEEE 50*, 3 (2012), 178–184.

[77] TANENBAUM, A. S. *Computer Networks*. Pearson Education, New Jersey, USA, 2012.

[78] VERMESAN, O., FRIESS, P., GUILLEMIN, P., GUSMEROLI, S., SUNDMAEKER, H., BASSI, A., JUBERT, I. S., MAZURA, M., HARRISON, M., EISENHAUER, M., ET AL. Internet of things strategic research roadmap. *Internet of Things: Global Technological and Societal Trends 1* (2011), 9–52.

[79] WANT, R. An introduction to rfid technology. *Pervasive Computing, IEEE 5*, 1 (2006), 25–33.

[80] WIFI-ALLIANCE. Wifi-Alliance homepage, 2016. URL: `http://www.wi-fi.org/`, accessed: 26.4.2016.

[81] WINDRIVER. Security in the internet of things, lessons from the past for the connected future. *Security Solutions, Wind River, White Paper* (2015).

[82] WINTER, T., THUBERT, P., BRANDT, A., HUI, J., KELSEY, R., LEVIS, P., PISTER, K., STRUIK, R., VASSEUR, J., AND ALEXANDER, R. Rpl: Ipv6 routing protocol for low-power and lossy networks. RFC-6550, March 2012.

[83] XI, W., ZHAO, J., LI, X. Y., ZHAO, K., TANG, S., LIU, X., AND JIANG, Z. Electronic frog eye: Counting crowd using wifi. In *INFOCOM, 2014 Proceedings IEEE* (April 2014), pp. 361–369.

[84] XU, L. D., HE, W., AND LI, S. Internet of things in industries: a survey. *Industrial Informatics, IEEE Transactions on 10*, 4 (2014), 2233–2243.

[85] Z-WAVE ALLIANCE. Z-Wave Homepage, 2016. URL: `http://z-wavealliance.org`, accessed: 13.1.2016.

[86] ZACHARIAH, T., KLUGMAN, N., CAMPBELL, B., ADKINS, J., JACKSON, N., AND DUTTA, P. The internet of things has a gateway problem. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications* (2015), ACM, pp. 27–32.

[87] ZHU, Q., WANG, R., CHEN, Q., LIU, Y., AND QIN, W. Iot gateway: Bridging wireless sensor networks into internet of things. In *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on* (2010), IEEE, pp. 347–352.

[88] ZIGBEE ALLIANCE. *ZigBee PRO Specification*, 2012.

[89] ZIGBEE ALLIANCE. *ZigBee IP Specification*, 2014.