FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Enabling the Blockchain in the Internet of Things

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Simon Krejci, BSc.
Matrikelnummer 01227059

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Assistant Prof. Dr.-Ing Stefan Schulte

Wien, 8. August 2018

Simon Krejci Stefan Schulte

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

TU WIEN
FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Enabling the Blockchain in the Internet of Things

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Simon Krejci, BSc.
Registration Number 01227059

to the Faculty of Informatics

at the TU Wien

Advisor: Assistant Prof. Dr.-Ing Stefan Schulte

Vienna, 8th August, 2018

Simon Krejci	Stefan Schulte

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Erklärung zur Verfassung der Arbeit

Simon Krejci, BSc.
Emil-Kralik-Gasse 4, 1050 Wien

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

Wien, 8. August 2018

____
Simon Krejci

Danksagung

Ich möchte die Gelegenheit nutzen, um jenen Menschen meinen Dank auszudrücken, die mich während meines Studiums und beim Schreiben dieser Diplomarbeit begleitet und unterstützt haben. An erster Stelle möchte ich meinem Betreuer Herrn Stefan Schulte für die regelmäßigen Treffen, das wertvolle Feedback und die gute Kommunikation während der Entstehung dieser Diplomarbeit danken.

Des Weiteren möchte ich meine Studienkollegen erwähnen, die mich seit den ersten Stunden an der TU Wien begleitet haben. Danke Daniel, Lukas und Lukas, dass wir die Herausforderungen des Studiums zusammen bewältigen konnten.

Danken möchte ich auch meiner Familie für ihren Rückhalt. Danke Yvonne, für deine stetige Unterstützung und die entspannenden Kaffeepausen. Danke Günter, für die Versorgung mit Spezialitäten aus der Heimat und die väterlichen Ratschläge.

Zu guter Letzt möchte ich Lena danken, die mich während der letzten Jahre begleitet, unterstützt und bei Herausforderungen angetrieben hat, mit 100% voranzugehen. Danke!

Kurzfassung

Die Blockchain und das Internet der Dinge (IoD) sind aufkommende Technologien. Die Blockchain kann als öffentliches, verteiltes Kontobuch beschrieben werden, welches auf einem Peer-to-Peer-Netzwerk aufgebaut ist. Das IoD ist eine Verknüpfung von Sensoren und Aktoren, in der Informationen über Plattformen hinweg ausgetauscht werden können. Im Allgemeinen kann man sagen, dass die Technologien für verschiedene Anwendungsbereiche vorgesehen sind. Allerdings hat die Blockchain das Potential, Probleme im IoD zu lösen. Trotz der Popularität hält sich die Anzahl der Versuche zur Integration der Blockchain im IoD in Grenzen. Darüber hinaus konzentrieren sich die bestehenden Ansätze auf grundlegende Fragen anstatt die Blockchain-Technologie in komplexeren Anwendungen zu verwenden. Aus diesem Grund, beschäftigt sich diese Diplomarbeit mit der Frage, wie die Blockchain aus softwaretechnischer Sicht im IoD verwendet werden kann.

Im Laufe dieser Arbeit entwickeln und implementieren wir die Blockchain-IoD-Anwendung, welche Ethereum verwendet. Die Anwendung kann Daten von Sensoren erfassen und über zwei Kommunikationskanäle verteilen. Im Mittelpunkt der Anwendung steht eine Middleware, die einen Sensortreiber, der die Sensordaten sammelt, mit Klienten verbindet, welche die verschiedenen Kommunikationskanäle abonnieren. Darüber hinaus ist der IoD Klient implementiert, der in der Lage ist, die Verzögerungen beider Kommunikationskanäle zu überwachen und die relevanten Informationen der Blockchain für die abschließende Auswertung zu verfolgen. Die entwickelte Anwendung wird schließlich auf drei verschiedenen Geräten evaluiert, nämlich auf einem Intel Galileo Gen2, einem Odroid-XU4 und einem Raspberry Pi 3.

Ein Ergebnis dieser Arbeit ist die Blockchain-IoD Anwendung und der IoD Klient. Das gesamte Framework ist in der Lage, Daten von Sensoren zu sammeln und über einen Echtzeitkanal und einen Kanal mit garantierter Integrität zu verteilen. Zusätzlich ist der IoD Klient in der Lage, die Verzögerungen beider Kanäle und die Bestätigungszeiten von Transaktionen in Ethereum zu überwachen. Unsere Ergebnisse zeigen, dass Ethereum auf dem Intel Galileo Gen2 nicht anwendbar ist. Die Einrichtung der entwickelten Anwendung auf den anderen Geräten funktioniert. Unterschiedliche Arbeitsauslastungen zeigen jedoch, dass die Portabilität der Anwendung eingeschränkt ist. Die beste Leistung in Bezug auf Verluste von Nachrichten in der Kommunikation erzielt der Raspberry Pi 3.

Abstract

The blockchain and the *Internet of Things* (IoT) are emerging technologies. The blockchain can be described as a public, distributed ledger, which is built on a *peer-to-peer* (P2P) network. The IoT is an interconnection of sensing and actuating devices where information can be shared across platforms. Generally, it can be said that these two technologies operate in different application fields. However, the blockchain has the potential to solve challenges of the IoT. Despite of the popularity, the amount of approaches to use the blockchain in the IoT is limited. Additionally, the existing approaches focus on basic questions instead of using the blockchain technology in more complex applications. Thus, this thesis deals with the question how to use the blockchain in the IoT from a software engineering perspective.

During the course of this thesis we design and implement the blockchain-IoT application which uses Ethereum. The application is able to collect data from sensors and distribute the data via two communication channels. The core of the application is a middleware which connects a sensor driver that collects the sensor data with clients that subscribe to the different communication channels. Furthermore, the IoT client is implemented which is able to monitor the delays of both communication channels and which tracks relevant information of the blockchain for the final evaluation. The developed application is evaluated on three different IoT devices, namely on an Intel Galileo Gen2, an Odroid-XU4 and a Raspberry Pi 3.

One result of this thesis is the blockchain-IoT application and the IoT client. The whole framework is able to collect data from sensors and distribute it via a real-time channel and a channel with guaranteed integrity. Additionally, the IoT client is able to monitor the delays of both channels and the confirmation times of transactions in Ethereum. The evaluation results show that enabling Ethereum on the Intel Galileo Gen2 is not feasible. The setup of the developed application on the other devices is possible. However, different workloads show that the portability of the application is restricted. The best performance in terms of message losses is achieved on the Raspberry Pi 3.

Contents

CHAPTER 1

Introduction

Motivation

According to Gartner's hype cycle [1] for emerging technologies in 2017, the IoT and the *blockchain* are currently at the peak of inflated expectations. Generally it can be said that these technologies are aiming at different application fields. Nevertheless, Dorri et al. [2] and Christidis et al. [3] discuss that combining these two technologies is promising for solving some challenges of the IoT. Especially the scalability and robustness of the ever-expanding IoT device ecosystem shall be improved by the blockchain's decentralized architecture. The decentralization avoids a single point of failure and decreases delays by eliminating many-to-one traffic flows. Additionally, the core functionality of the blockchain as a digital ledger which stores in case of cryptocurrencies all money transactions is beneficial. It can be used to conveniently add a billing layer to the IoT infrastructure. Assuring secure money transfers demands for sophisticated cryptography and security concepts like they are provided by the blockchain. Hence, this high standard of security can also be of use for the IoT. Because of the named advantages, it makes sense to apply these two technologies complementarily.

Besides the scientific point of view, combining the blockchain and IoT can also be interesting for real world scenarios. An example is the regulation on a mechanism for monitoring and reporting greenhouse gas emissions of the European Union [4]. Here, environmental information is collected at national and Union level to track progress in reducing emissions which contribute to climate change. Therefore, the quality, immutability and consistency of data is important. Currently the data is transmitted to the European Commission in form of an annual report. Implementing the blockchain and IoT in this scenario could make this process more sophisticated. The environmental data could be collected by an IoT sensor network which stores the data directly into the blockchain. Primarily, this would guarantee tamper protection because once a transaction is added to the blockchain it can not be changed any more. Another advantage is that the measured values would be immediately accessible and the European Commission could monitor data in nearly real-time without waiting for the annual report. However, before doing so, apart from the integration of these technologies, different issues have to be solved.

Aim of the Work

Despite the promising application of the blockchain in IoT, there are some challenges to overcome. Dorri et al. [5] name resource restrictions, considering computational power and bandwidth as a critical bottleneck of IoT devices. Furthermore, the poor scalability of the blockchain, when the network contains too many nodes, is a drawback. This is a real challenge because IoT networks are expected to have numerous nodes. Another issue is that storing sensor data in the blockchain will bloat it with transactions and data [6]. Because of the underlying P2P architecture, every node in the network is storing the transactions and data. Thus, finding a strategy which decreases the data loads on the blockchain is a challenge. Regarding the research community, the publications are not that numerous and diverse as it might be expected from the current popularity of the topic. A systematic literature review from Conoscenti et al. [7] in 2016 identifies four use cases and five papers about combining the blockchain and the IoT. Futhermore, the majority of blockchain research concentrates on Bitcoin [8]. However, the research community seems to be engaged in these topics and continuously publishes new results.

The impression from the current publications is that a lot of papers deal with the topic blockchain and IoT on a low level. This means that the papers focus on basic questions instead of using the blockchain in more complex applications. Thus, the question rises how the benefits of the blockchain can be used in an IoT application. Furthermore, blockchain technologies need a lot of resources. Despite the existence of possiblities to access the blockchain with lower resource usage, the question remains how does it perform on resource-restricted devices. Therefore, another question is how to enable the blockchain on resource-restricted devices and how does it perform. This thesis focuses on the following topics:

Application Design

As it has already been mentioned, the research aiming at specific use cases is rather rare. Furthermore, research in this domain focuses more on technical details than on software engineering problems. Hence, the aim of this work is to develop an IoT application which fits into the blockchain-IoT domain. Therefore, the application implements a use case which is inspired by the scenario about the monitoring and reporting of greenhouse gas emissions. Furthermore, sensors are a basic element of the IoT [6]. Thus, the application has to be able to operate with sensors to fit in the IoT. The data which is generated by the sensors has to be handled in the application. The resulting application is then included in the final performance tests. Regarding the implementation, the core of the blockchain-IoT application is a middleware. A middleware is an approach to hide the complexity of an underlying infrastructure [9]. This software abstraction layer helps

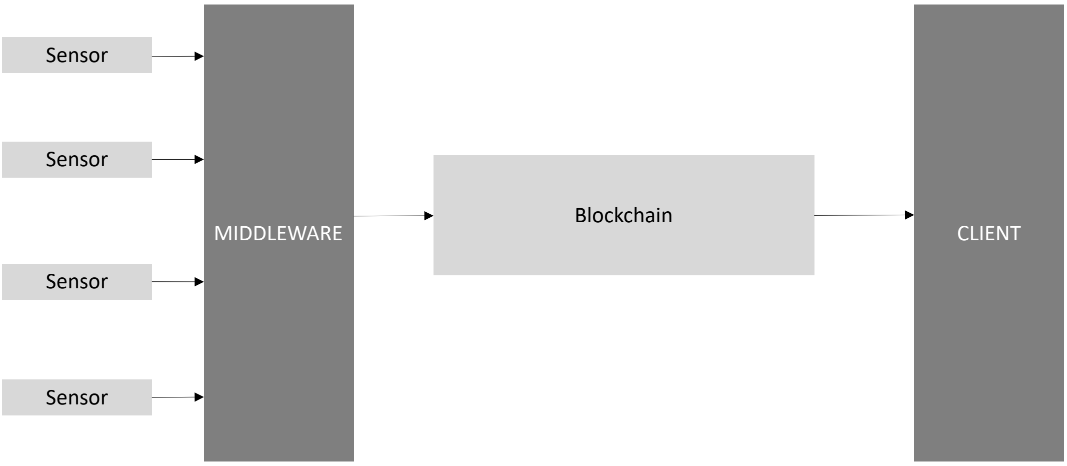


Figure 1.1: Sketch of Solution Approach.

a developer to focus on his tasks without considering the underlying complexity. In case of the work's blockchain-IoT application, blockchain access and data collection can be hidden. The service requirements for a complete middleware in the IoT are resource discovery, resource management, data management, event management and code management [9]. Nevertheless, the aim of this work is not to implement a complete IoT middleware. The aim is to implement a middleware which contains data management functionalities and enhances it with the blockchain technology. The approach is sketched in Figure 1.1. Data is collected by the middleware from the connected sensors and distributed via the blockchain. To retrieve the data, a client has to be connected with the blockchain.

One important aspect of data management is context awareness. The issue with data in IoT is the heterogenity of IoT devices and their data [9]. When a value is received from a device, it can not be said if it describes the temperature or humidity without further information. Storing this additional information, like type of measurement, enables a context-aware handling of the measured values and therefore a targeted use. Thus, the consideration of the context awareness is part of the development process.

Performance

As already has been explained, performance is critical because of the resource restrictions of IoT devices. Besides the general duties of the IoT devices, which already consume computing power, a client program must be executed to access a blockchain. This means extra computations which may be a problem for the devices. Therefore, the aim is to find a strategy to enable the blockchain or rather the client on the IoT device. However, besides the blockchain, the IoT device shall also be able to run an application and provide services for clients. With this setup, the performance is tested on different IoT devices. A final evaluation will show the devices' suitability for the participation in the blockchain network and the execution of an IoT application.

To sum up, the aim of this work is to enable the blockchain on IoT devices. For the performance evaluation on different devices, an application is developed to test the blockchain in action. This application aims to implement the monitoring use case of Section 1. We expect that this work will show the feasibility of the blockchain on resource-restricted devices. Furthermore, the development of the blockchain-IoT application shall reveal how advantages of the blockchain can be used in the middleware data management.

Methodology and Approach

The results of this work are approached in several steps. First of all, a literature research is performed, to develop an understanding of blockchains, IoT and how to combine those two technologies. Furthermore, the literature research shows the state-of-the-art about current solutions and challenges. The literature research is methodologically based on the guidelines for systematic reviews and empirical research in software engineering by Kitchenham et al. [10]. With the results of the literature research, the functional and nonfunctional requirements for the blockchain-IoT application are defined. Based on the requirements, the architecture is developed. Furthermore, a client is developed which accesses the services of the middleware whereas the client is used to test and measure the functionality.

After the implementation phase, the developed blockchain-IoT application is installed with its required components on three different IoT devices. The three used devices are the Intel Galileo Gen2, Raspberry Pi 3 Model B and ODROID-XU4. The motivation behind this selection is to have devices with three different performance levels. The ODROID-XU4 is the strongest device, the Raspberry Pi 3 Model B the second strongest and the Intel Galileo Gen2 the weakest one. The goal of this phase is to test if it is possible to setup the devices such that the blockchain-IoT application can be executed.

The final step is to evaluate the blockchain-IoT application and the devices where the setup is successful. One part of the performance evaluation concentrates on the performance of the IoT device itself like *central processing unit* (CPU) or memory utilization. The second part focuses on the performance of the blockchain-IoT application. The metrics for the blockchain-IoT application are determined in the design and development process.

Structure

The thesis is structured as follows. Chapter 2 presents basic information about the blockchain and IoT. This chapter provides the introduction for the reader to the topics which form the basis for the following chapters. Chapter 3 gives a deeper insight into specific topics which are relevant for this work and for the solution approach. Therefore, current solutions which combine the blockchain and IoT are described. Additionally, different blockchains, storage possibilities and communication protocols, which are likely to be used in the blockchain-IoT application, are discussed. Chapter 2 and 3 are based on the results of the literature research. The following Chapter 4 deals with the requirements analysis for the blockchain-IoT application and presents the design for the implementation. The implementation itself is documented in Chapter 5. Chapter 6 documents the performance evaluation approach. Furthermore, the results of the evaluation are presented. Finally, Chapter 7 summarizes the work and gives an outlook for future work to answer the question which raised during the current work.

CHAPTER 2

Background

This chapter provides elementary concepts and the theoretical background for the topics which form the basis of this Diploma Thesis. To follow a structured approach for the literature research, Section 2.1 defines the review protocol. Section 2.2 introduces the blockchain domain. Using the example of Bitcoin, basic concepts and the architecture of blockchains are explained. Furthermore, next-generation technologies, like smart contracts and alternative consensus algorithms, are part of this overview. Section 2.3 focuses on IoT topics like architecture, elements of this network and challenges. In addition, middleware approaches are discussed.

2.1 Systematic Literature Review

A literature review is used to create the content of the chapters “Background” and “Related Work”. To improve the quality of the review by approaching it in a systematic and structured way, we use the *systematic literature review* (SLR) approach of Kitchenham [10] as orientation. An SLR is basically a secondary study, which answers specific research questions in a certain field of interest. It follows a well-defined methodology, which assures repeatability and transparency. This well-defined methodology causes a lot of overload, which would be too much for the introductory Chapters 2 and 3. Therefore, we adapt the approach in a way that the overload is reduced to a necessary minimum and the repeatability is increased at the same time.

2.1.1 Review Protocol

For the structure and to avoid a bias on the study selection, a review protocol must be defined before starting with the actual research. The protocol for this thesis is defined as follows:

(1) **Background:** The core topics of this thesis are blockchain technologies and the IoT. Additionally, it is tried to combine both technologies and achieve an improvement of specific use cases. So the two chapters "Background" and "Related Work" shall provide an introduction and a basic overview to the current state-of-the-art in these fields. The aim of the background chapter is to explain basics like terminology and principles in the blockchain and IoT domain. The related work chapter focuses on current solutions in terms of combining those two technologies.

(2) **Research Questions:** The aim of this SLR is to provide an overview. Thus, the questions will not be as specific as they would be in other SLRs. Also the answers will not claim completeness in terms of questions like which approaches do exist. Nevertheless, they still provide a good structure for the research process. Because we will discuss various topics in these chapters, three categories of research questions are introduced. The abbreviation BQ is used for those which will be answered in the background chapter for the blockchain and IoT domain. Specific blockchain topics are indicated by BBQ. RWQ is used for the definition of specific questions in Chapter 3.

(BQ1) What is the core of this domain? Explain basics, terminology and principles.

(BQ2) What are the current challenges?

(BBQ1) What are smart contracts?

(3) **Research Strategy:** In an SLR as described by Kitchenham, the research strategy must be well-defined, and every step has to be documented. Defining study selection criteria, quality assessment and data extraction strategies causes a lot of overhead. To keep the research for the introduction chapters of this thesis within an appropriate extend, these points will be held on a minimum.

For the search, Google Scholar[1] is used. The results are mainly taken from well-known digital libraries, e.g. IEEE[2], ACM[3], Springer[4] and Elsevier[5]. Also, the recommendation service of Mendeley[6] supports the research. This reference management tool sends you now and then a list of suggestions with similar papers to them, which are already in your Mendeley library. These suggestions will be reviewed. In case they provide content to answer the defined research questions, they are added to the bibliography. In the "Related Work" part actual blockchains are presented. Therefore, the websites and whitepapers of the certain blockchains are accepted as valid sources.

In this review, also secondary studies like surveys are allowed. The reason for that is, they provide the content to get an overview about the topics. To follow the guidelines

[1] https://scholar.google.at [Accessed: 2018-02-10]
[2] http://ieeexplore.ieee.org [Accessed: 2018-02-10]
[3] https://dl.acm.org [Accessed: 2018-02-10]
[4] https://link.springer.com [Accessed: 2018-02-10]
[5] https://www.elsevier.com [Accessed: 2018-02-10]
[6] https://www.mendeley.com [Accessed: 2018-02-10]

of Kitchenham, the quality assessment comes after the study selection. Most of the quality is assured by using well-known digital libraries. According to papers with technical solutions, it will be checked if there are actual implementations provided by the authors. The papers with existing implementations are rated with a higher quality. For the data extraction no extra forms are used. The results are immediately written down in the Chapters 2 and 3, which is at the same time the form in which these results are disseminated. The extracted data will be reviewed by the advisor of this thesis.

2.2 Blockchain

The blockchain came up in 2009 with the digital money Bitcoin proposed by Satoshi Nakamoto [11]. Alltough the idea of digital money has already existed since the early 1980s [12], it took more than twenty years to develop a fully distributed solution in form of Bitcoin and its underlying blockchain technology. The basic idea of the blockchain is to get rid of a central authority. In terms of money transactions, banks are in the role of a central authority these days. They are supervising the transactions and guarantee their validity. On the one hand this creates security, on the other hand you have to trust in the banks' integrity. Another example for central authorities only for data are social networks. According to Zyskind et al. [13], Facebook has gathered 300 petabytes of personal data from the launch of the platform in 2004 [14] till 2014. Generally, it can be said that we are living in a Big Data era, where intelligence is permanently collected, analysed and used to improve companies' services and profits [13]. That means, others control your personal data and you have to trust that the data is used in the way you want it to be used.

With its P2P architecture and the use of cryptography, a blockchain is able to avoid the central authority. Originally it was developed to execute money transactions between two parties without a trusted third party, but the concept can be extended and used for other application scenarios like the protection of personal data [13].

2.2.1 Concepts and Terminology

Bitcoin was the first blockchain and was deployed in 2009. Hence, we present an introduction to the blockchain based on the example of Bitcoin. The content for this section relates to the work of Tschorsch et al., Xu et al. and Zheng et al. [12, 15, 16].

Generally, a blockchain can be described as a public, distributed ledger, which is built on a P2P network where security is achieved by cryptography. All transactions, which are executed in this network, are recorded on this ledger and stored on every node in the P2P network. Basically, the ledger looks as depicted in Figure 2.1. As it can be seen several blocks are linked to each other like a chain of blocks. This data structure is the reason for the name blockchain.

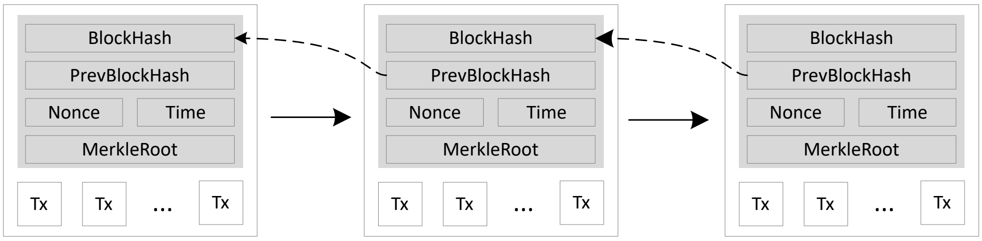


Figure 2.1: Simplified Example of a Blockchain. (Source: [12])

Each block comprises a number of transactions. The transactions can be seen as an abstract representation of coins in the blockchain. Every transaction has a hash to identify it and a list of inputs and outputs. The input list is used to reference outputs of previous transactions. However, an output can only be used as input once in the whole chain. In case the output is used twice this would be like spending a coin twice, and this is not allowed. Therefore, the transactions can be divided into the two categories *unspent transaction output* (UTXO) and *spent transaction output* (STXO). An UTXO is not referenced by a following transaction input at the moment. Due to the linking of the transactions, the history of transactions can be traced back. The origin of every transaction history is either the *genesis block*, which is the first block of the chain, or the *coinbase transaction*. The coinbase transaction has only outputs and is used to rewarded miners. Thus, coins are added to the system. Since for standard transactions the sum of all inputs have to be equal to the sum of all outputs, coinbase transactions are the only possibility to bring coins into the system. The genesis block supplies only once coins to the system.

Besides the transactions, a block also contains the header which comprises several elements. First, the *BlockHash* is the hash value of the block and the *PrevBlockHash* refers to the BlockHash of the previous block. With the PrevBlockHash as a pointer the data structure of the chain is like a linked list. Therefore, the order of the blocks and transactions can be determined. Since the order is determined, also the current ownership of a coin is determined. Furthermore, the blockchain is tamper-proof since the hash depends on the block's content. If something changes, the hash changes too and thus the references are not correct anymore, which results in an invalid chain. Thus, manipulations can not be done without recognition by other peers.

The next element is the *nonce* which is needed for the security of the system. To achieve an infrastructure which gets along without a third party, the *double spending problem* and the *Sybil attack* threat have to be solved. Spending a physical coin twice is not possible. That seems trivial in the real world is not that easy with digital money in a distributed environment. In a cryptocurrency, the double spending problem can be controlled by consensus of the nodes in the network. To verify a transaction positively, the majority has to agree on the correctness of the transaction. This only works under the assumption that every node is honest. A dishonest node could start multiple instances to manipulate the consensus vote (Sybil attack). To avoid this, nodes have to verify the

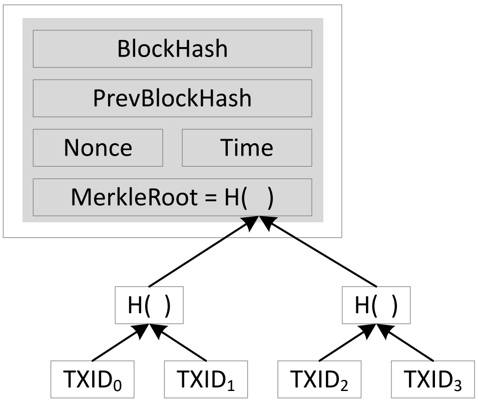


Figure 2.2: Merkle Root. (Source: [12])

transactions using *proof-of-work* (PoW). The assumption of the PoW is that it is harder for an attacker to dominate the majority of the network's computing power than the majority of instances. However, recent research by Eyal et al. [17] shows that only one third of total computing power is needed to attack the blockchain.

When a new transaction is triggered, it is broadcast to the whole network. The nodes verify the transaction. After that, the transaction is collected in a block together with a number of other transactions. To generate the new block, which is called *mining*, a computationally intensive puzzle has to be solved namely the PoW. The puzzle is solved by calculating a hash of the block including the header and the transactions. The nonce in the header is adjusted until the hash is equal to or lower than a specific target value. Finally, the block with the correct hash is distributed in the network and added to the blockchain by all nodes. Sometimes it can happen that two canonical blocks are mined at the same time. In this case the blockchain forks and different versions of the blockchain do exist. Thus, there might not be a consensus on the order of the transactions in the network. The solution of Bitcoin is to continue mining on the longest fork whereas "longest" means the fork where most of the computing power is involved [12]. Miners decide locally which fork is the longest and continue mining on this fork and broadcast the new mined block. Thus, one fork overtakes other forks and is then called the main chain. Blocks which are not in the main chain are lost and subsequently the contained transactions.

The last element in the header is the *MerkleRoot* which is the root node of a *Merkle tree* [18]. Figure 2.2 shows the Merkle tree where hashes are calculated pairwise from the leaf nodes. At higher levels, hashes are calculated pairwise from hashes. Hence, it can be checked if a transaction is contained in the block or not by checking the root hash. The MerkleRoot eases scalability issues of the blockchain because thin clients can be realized. In contrast to full nodes, which store all blocks and their transactions, thin clients only store the headers. To verify if a transaction is stored in the block, the transactions for the pertaining block have to be downloaded. Then the Merkle tree can be built. If the root hashes are equal, the transaction is in the block.

2.2.2 Smart Contracts

A *smart contract* expands the functionality of the blockchain. Christidis et al. [3] describe a smart contract as a script which is stored in the blockchain and can be accessed by its own address. Its behavior is completely deterministic. When a transaction is sent to the contract's address, the contract is triggered. With these contracts, proper applications can be implemented in the blockchain. A blockchain, which provides this functionality, is the Ethereum blockchain [19]. In Ethereum, smart contracts run on an own virtual state machine which is called the *Ethereum Virtual Machine* (EVM). This machine is quasi-Turing-complete because a parameter limits the number of computations. Besides that, there are no limitations considering scripting.

One problem with smart contracts is the data storage because blockchains are not designed to persist a lot of data [20]. The second problem is the reachability of external data because smart contracts have only access to data within the blockchain environment. However, access to external data is necessary for real-world scenarios. A possible use case is a flight delay insurance. To let a smart contract decide if a flight is delayed, it must know the actual departure and arrival times. Hence, external data is essential to provide a sound application. Therefore, developers use so-called *oracles*. These oracles are external services which feed smart contracts with data. Such services are already existent. For example, Zhang et al. [21] present an oracle that bridges smart contracts and websites.

2.2.3 Consensus Algorithms

The PoW concept, like it is used in Bitcoin, is completely dependent on powerful hardware which consumes a lot of energy. Although there exists special hardware, which is called *application-specific integrated circuit* (ASIC), to find hashes more efficiently, the energy consumption is still high. According to O'Dwyer and Malone [22], the total amount of electricity which was used by the Bitcoin blockchain from January 2009 till January 2014 is approximately ten gigawatts. To get an impression about this value, it can be compared to the power of one billion LED lamps [23]. Also quite impressive is that it is a tenth of the United States of America's solar photovoltaic and wind power (Oct. 2015). The thing about the ten gigawatts is that is has been the consumption from the very beginning. In the first year the consumption was nearly constant. Since the second year, the consumption is rising and the trend is still going upwards [24].

Because of this huge waste of energy and the need for special hardware, other approaches than PoW are appearing in the community. These approaches try to get rid of the dependency on powerful hardware devices, to save energy and to avoid a centralization of the computing power by a few rich stakeholders.

Proof of Stake

The aim of the *proof-of-stake* (PoS) approach is to secure the P2P network without the dependency on energy consumption [25]. To establish security, it follows a proof of ownership approach where the ownership is related to the coin age. The coin age is defined as the product of the amount of coins and their holding period. For instance, if Alice receives ten coins and holds them for ten days then the coin age is 100 coin-days. To generate a new block, a hash target has to be found like in the PoW, but the difference is that this proof has a limited search space. Thus, the PoS does not consume a significant amount of energy. The proof is designed in a way, the more coin age is consumed in a block the easier it is to meet the hash function. This can be related to the computing power in the PoW where it is more likely to find the target hash the more computing power is available. Because of the coin age, the stake is not only depended on the amount of money but also on the holding period. In a nutshell, it can be said that the user with a higher stake is more likely to generate a new block. In case there are two competing chains, because of a conflict, the chain with the most consumed coin age wins.

In comparison to the Bitcoin's PoW, the PoS is able to provide some security improvements [25]. At first, it might be more expensive to acquire the significant fraction of stake than acquiring enough mining power to reach the majority in the network. Additionally, an attacker's coin age is consumed during the attack. Hence, it will be harder for the attacker to tamper the chain by providing a stake which is big enough.

Proof of Activity

Bentov et al. [26] describe *proof-of-activity* (PoA) approach as a combination of PoW and PoS. The motivation is to make attacks more expensive by forcing the attacker to dominate computing power and the stake. Another reason for this approach is that PoW, according to the authors, has not the right incentive to secure the network. For instance, if the earnings from a block generation is only accumulated by transaction fees, then the motivation to invest more money into mining hardware and therefore secure the network will drop. This issue is well known in economic and game theory under the name "Tragedy of the Commons" by Hardin [27]. Centralization is another problem by PoW and PoS. In terms of the computing power, huge data centers are able to outperform private miners easily. Although PoS reduces the risk of computation power accumulation, wealthy stakeholders can build enormous stakes. PoA aims to solve those issues.

For the protocol, a subroutine, called *follow-the-satoshi*, is used. This routine takes a pseudorandom value as input and selects a *satoshi* with a pseudorandom index. A satoshi is the smallest unit of cryptocurrency in Bitcoin. This selected satoshi is followed from the block where it was minted until the current block. Thus, the owner of the coin can be determined in the current block and therefore a pseudorandom stakeholder is found. This procedure is a variant of a PoS. The only difference is that the stake relates to coins instead of coin age. For example if Alice has four coins and Bob has eight coins, Bob is double likely to be picked than Alice.

At the beginning of the protocol each miner tries to generate an empty block by finding the correct hash. If a miner generates a block, he broadcasts it to the whole network. From this block, N pseudorandom stakeholders are derived. This is done by generating N hashes and invoking the follow-the-satoshi subroutine with each of the hashes. Every online stakeholder checks the validity of the block. During this validation, he can find out if he is one of the N chosen stakeholders. If this is the case, he signs the block with his private key and broadcasts it. This is done by $N-1$ of the chosen stakeholders. The N^{th} stakeholder generates a wrapping block where he adds the empty block, arbitrary many transactions, the signature of the other $N-1$ stakeholders and his own signature. After that, the block is broadcast. When the other nodes agree on the validity of the block, it is added to the blockchain. The fees of the transactions are shared between the miners and the N stakeholders.

Hence, by selecting N pseudorandom stakeholders, the validation of the transactions is decentralized. Furthermore, the fees are shared which shall engage the community to be active in the network and therefore improve the security.

Proof of Burn

In the whitepaper of Slimcoin [28] *proof-of-burn* (PoB) is described . Like the other approaches, it also aims to get away from the dependency on powerful hardware and it also combines PoS and PoW. Therefore, it follows the idea of burning coins. Burning coins can be compared to purchasing hardware for the PoW. So to speak the more coins are burnt the more computing power you have. In the blockchain, this destruction is realized with a special transaction. By triggering the so-called *burn transaction*, coins are sent to an address which has no owner. Thus, these coins can not be used any more except for generating new blocks. All burn transactions are recorded separately from the other transactions in the system. Nevertheless, they also have to gain some depth in the blockchain to be seen as confirmed. When they are confirmed, *burn hashes* can be calculated. These burn hashes are the same as the best solution of a hash computation in the PoW. In the Slimcoin paper, the burn hash is calculated as Equation 2.1 shows.

$$BurnHash = multiplier * InternalHash \tag{2.1}$$

For each burn transaction, there is a different multiplier. This multiplier is inversely proportional to the burnt coins. Because of that, a decay of the coins can be realized. For instance, if a transaction burns 60 coins and the multiplier triples in 100 days then there exists the same multiplier for 20 coins on the 100^{th} day. The second part of the equation is the internal hash. The hash is calculated only when a new PoW block is found. This applies also to the burn hashes. Therefore, the PoB does not consume that much energy. Another energy-saving method is the limitation of the search space for PoB hashes, like in PoS.

To summarize, the PoB concept simulates the purchase of new hardware equipment by burning coins. The more coins are burnt, the higher is the probability of mining a block successfully. Furthermore, over time more and more coins have to be burnt because the burn hash calculation contains a sort of decay. Thus, every miner must continue burning to maintain the probability to mine a block. Therefore, the volume of burnt coins increases and it is getting harder for an attacker to dominate more than 50% of the volume, which makes the network more secure.

2.2.4 Challenges

This section gives an overview about the challenges which have to be faced when dealing with the blockchain domain. Swan [29] identifies seven technical problems. To find out the current research state of these problems, Yli-Huumo et al. [8] perform a systematic review. The presentation of the seven technical problems, commented with the results of the review from Yli-Huumo et al., is content of this section. A result of this review is that 80% of the papers focus on Bitcoin. Furthermore, it can be said that Yli-Huumo et al. identify no research papers for the topics throughput, latency, size, bandwidth, versioning, hard forks and multiple chains.

Throughput: The throughput in the Bitcoin network has a maximum of seven *transactions per second* (tps). Comparing this to other systems, like VISA with up to 10,000 tps or Twitter with a maximum of 15,000 tps, it is rather slow. To provide a real alternative to current currency systems, the performance has to be improved. Projects like the Bitcoin Lightning Network [30] or Ethereum's sharding approach [31] are working on improving the throughput or rather the scalability in general.

Latency: Processing a block in Bitcoin takes approximately ten minutes. In the Ethereum blockchain it takes a lot less time, namely twelve seconds [32]. However, additionally to the processing effort some time has to pass to assure that the block is in the main chain. In Bitcoin the duration is about an hour for a transaction. In comparison, the confirmation with VISA only takes seconds.

Size and bandwidth: The size of the Bitcoin is nearly 159 GB (Feb. 2018) [33]. Allthough Ethereum provides a fast synchronization mode besides the full synchronization, it still has 51 GB (Feb. 2018) [34]. If the throughput increases to the VISA average of 2,000 tps in Bitcoin, the annual growth is 1.42 PB or rather 3.9 GB/day. With this data loads it is hard to make the blockchain feasible for the mainstream. Also conquering new domains like the IoT with its resource-restricted devices is a challenge.

Security: One big threat to security is the 51% attack. According to recent research results by Eyal et al. [17] even a third is enough, to make the network insecure. Another issue is that mining methods tend to centralization. The importance of security is reflected by the results of the review because it is the most researched challenge. The topics are about trends and impacts of security incidents, the 51% attack, data malleability problems, authentication and cryptography issues.

Wasted resources: As has already been stated in section 2.2.3, mining consumes a lot of energy, which also costs a lot of money. However, the systematic review from

Yli-Huumo et al. reveals that energy efficiency is not a handled problem. There are only some papers which deal with topics related to wasted resources. For instance, Wang and Liu [35] explore the evolution of miners in Bitcoin. Furthermore, an economic model is introduced which calculates the miners' profits by considering electricity prices and the computation-over-power efficiency of the hardware.

Usability: The usability of the Bitcoin *Application Programming Interface* (API) is not user-friendly compared to modern APIs. The found literature, which relates to this topic, concentrates on the usability issues for cryptocurrency users. Usability for developers is not considered.

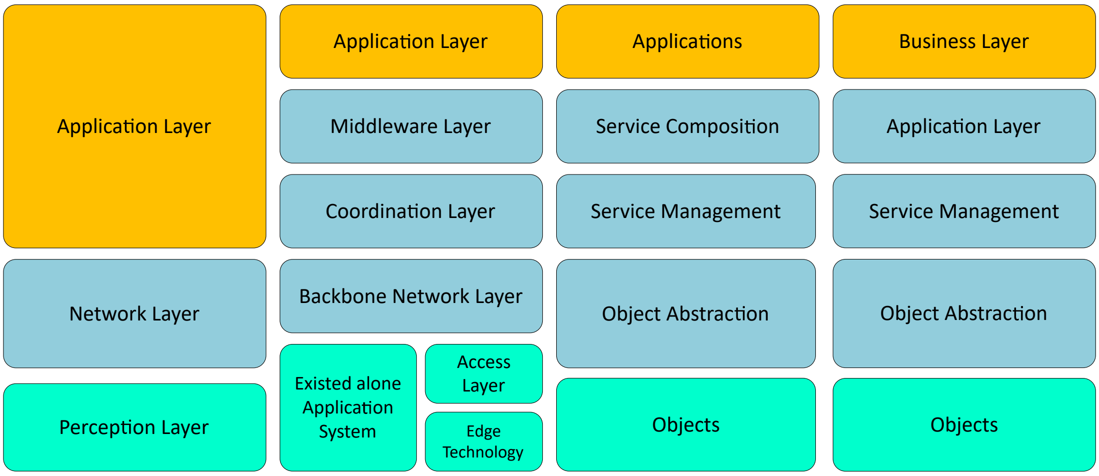
Versioning, hard forks, multiple chains: The large number of different blockchains is a threat because it is easier to execute a 51% attack on smaller chains. Also forking chains due to administrative reasons is a challenge, since there are no solutions to easily merge or cross-transact between the split chains.

2.3 Internet of Things

This section is mainly a summary of Al-Fuqaha et al. and Xu et al. [36, 37], except the middleware subsection. For IoT, a classical application area is the food supply chain. The current processes in the supply chain are complex and extremely distributed. Therefore, quality management, operational efficiency and food safety are very hard to manage. The IoT can help to deal with traceability, visibility and controllability challenges. A typical solution for this has three parts. The first part are the field devices which are used to collect the data. Furthermore, a backbone system and a communication infrastructure are needed. With this solution, it is possible to track and monitor food production. Additional technologies for Big Data help to analyze huge amounts of data which is generated along the whole food supply chain. On this example it can be seen that IoT is about enhancing physical objects, like food products, in a way that they can be identified and integrated in a digital infrastructure. Hence IoT can be defined as

> *Interconnection of sensing and actuating devices providing the ability to share information across platforms through a unified framework, developing a common operating picture for enabling innovative applications. This is achieved by seamless large scale sensing, data analytics and information representation using cutting edge ubiquitous sensing and cloud computing.[38]*

The rest of this section is organized as follows: Subsections 2.3.1 and 2.3.2 present different architectures and according elements for different architecture layers. The middleware Subsection 2.3.3 takes a closer look on middleware solutions in IoT. Finally basic challenges are presented in Subsection 2.3.4.



(a) Three-layer (b) Middleware-based (c) SOA-based (d) Five-layer

Figure 2.3: Different IoT architectures. (Source: [36])

2.3.1 Architecture

The IoT should be able to connect billions or trillions of devices trough the Internet [36]. Therefore, architectures must be flexible to cope with different requirements for various use cases. There exists a number of different architectures which have not converged to a reference model yet. Some of these approaches are shown in Figure 2.3. The basic model is the three-layer architecture (2.3a). Other models (2.3b, 2.3c, 2.3d,) have added more abstraction layers to the IoT architecture. In this section we will discuss the five-layer model in more detail.

Objects Layer: The layer is also called perception layer. Here, the data, for example temperature or humidity measurements, is collected with special hardware. Sensors and actuators are the two types of hardware which are used. This layer digitizes the data and transfers it to the Object Abstraction layer.

Object Abstraction Layer: The only function of this layer is to pass the data from the Objects Layer to the Service Management Layer. In case cloud computing or specific data management processes are required, it is handled by the Object Abstraction Layer.

Service Management Layer: It can also be seen as a middleware. The abstraction in this layer enables programmers to work with different objects without considering hardware details. Furthermore, it pairs services and their requesters according to the address and name. In addition, this layer processes data, makes decisions and delivers services.

Application Layer: In this layer, high-quality smart services are provided for the customers. For instance, in a smart home application temperature data can be delivered when a customers requests it. Thus, it can be seen as interface to the data which is collected by the objects.

Business Layer: On the highest level of the five-layer architecture all the activities and services of the IoT system are managed. It uses the Application Layer as interface to fetch the data and processes it to charts or business models. Hence, it can be used to support decision making. Furthermore, it monitors and manages the underlying four layers. Thus, basically the Business Layer unites all the components of the network and prepares it for the end user.

2.3.2 Elements

Several elements are needed, to build an IoT network. The six elements identification, sensing, communication, computation, services and semantics are the basic building blocks. In the rest of this section these blocks are presented. After that, Table 2.1 provides a summary of the presented elements and their according technologies.

Identification: It is key, to clearly identify objects in the network and match services to them. In the IoT, names and addresses are used for identification. Naming refers to the ID or rather the name of an object whereas addressing is the identification of an object within a network. Because of the different naming methods, like electronic product codes or ubiquitous codes, which are not globally unique, addressing is used to uniquely identify the objects.

Sensing: This is the very basis of the IoT. Objects collect all the data which is used for further processing. Smart sensors, actuators or wearable sensing devices can be objects. In comparison to sensors which sense the environment, actuators can affect the environment [39]. The affection can be done by making a sound or flashing a light. Often, sensors are used together with actuators. For instance, the sensor monitors the air in the room. In case of a dangerous gas in the air, the actuator emits a loud sound. Typical devices to realise an IoT product are single board computers with integrated sensors, like a Raspberry Pi [36]. The advantages of these devices is that they combine sensing, communication, and security functionalities on one board.

Communication: The connection of heterogeneous devices to form smart services is an elementary part of the IoT. An important aspect for the communication technologies is that they are able to cope with lossy and noisy links. At the same time, the power usage has to be low, to enable it also on resource-restricted nodes. The first technology, which was used for the communication in the *machine-to-machine* (M2M) concept, is the radio-frequency identification (RFID) [36]. Other technologies followed like *near field communication* (NFC), WiFi or Bluetooth.

Computation: Gathering data by sensors is only one part. It also needs processing units and software applications to use the collected data intelligently. Thus, special hardware and software is used to solve these tasks. In terms of software, operating systems play a key role in the IoT network because they run on the devices for the whole activation time. Also when it comes to measurements like temperature, it is sensible to provide them in real-time at least for some use cases. Therefore, special real-time operating systems are

Table 2.1: Summary of the IoT Elements and According Technologies. (Source: [36])

IoT Elements		**Samples**
Identification	**Naming**	EPC, uCode
	Addressing	IPv4, IPv6
Sensing		Smart Sensors, Wearable Sensing Devices, Embedded Sensors, Actuators, RFID Tag
Communication		RFID, NFC, UWB, Bluetooth, BLE, IEEE 802.15.4, Z-Wave, WiFi, WiFiDirect, LTE-A
Computation	**Hardware**	SmartThings, Arduino, Phidgets, Intel Galileo, Raspberry Pi, Gadgeteer, BeagleBone, Cubieboard, Smart Phones
	Software	OS (Contiki, TinyOS, LiteOS, Riot OS, Android); Cloud (Nimbits, Hadoop, etc.)
Service		Identity-related, Information Aggregation, Collaborative-Aware, Ubiquitous
Semantic		RDF, OWL, EXI

developed. Besides that, clouds are also a rich resource for computing power. Therefore, smart objects can send their data to the cloud where it is processed in real-time.

Services: A typical way to provide the collected IoT data and to process user interactions are services. These services can be split into four categories namely Identity-Related, Information Aggregation, Collaborative-Aware and Ubiquitous Services. The most essential services are the Identity-Related services because they bring the real world objects into the digital world by assigning an identity to the objects [36]. After that, these objects are in the system and can be tracked. The second category is the Information Aggregation Service. As the name says, it summarizes sensor data. Based on the aggregated data, Collaborative-Aware Services make decisions and trigger reactions. The fourth category are the Ubiquitous Services which aim to make Collaborative-Aware Services as accessible as possible independent from time, persons or places.

Semantics: For this element, Semantic Web technologies, for example Web Ontology Language (OWL) or Resource Description Framework (RDF), are used. These technologies are important to extract knowledge smartly from the network. Additionally, they analyze the data to provide the right services.

2.3.3 Middleware

Ngu et al. [40] name middlewares as key technology for the IoT. Razzaque et al. [9] argue that the structure of the IoT demands for middleware solutions to ease the development process. To specify the requirements of a middleware, it is good to know the characteristics of the IoT. First of all, the IoT is an ultra-large-scale network where a lot of interacting devices are connected. Thus, the number of events which are exchanged is also very high. Some of these events can be triggered spontaneously when, for instance, mobile objects come into the communication range of other objects. To keep track of the generated events and data, context-awareness is key. When a value is received from a device, it can not be said, if the value describes the temperature or humidity without further information. Storing this additional information enables a context-aware handling of the measured values and therefore a sensible use is possible. According to these characteristics, Razzaque et al. derived five functional requirements which have to be fulfilled by an IoT middleware. The five requirements are resource discovery, resource management, data management, event management and code management. Besides the functional requirements, also non-functional requirements are defined, for instance, scalability, security and privacy, reliability and availability.

Middleware in IoT is currently an active research area. Therefore, many solutions are implemented. According to Ngu et al. [40], these solutions can be divided into three types. The first type is the *service-based* IoT middleware which is a three-layered architecture. All the services like access control or event processing engines are located in the middle layer. The architecture is high-performing, however, the provided tools are generally simple. Furthermore, access restrictions can be set up but it is not designed to allow user configurations. To achieve the needed computing power, the service-based middleware is deployed in the cloud or on servers. Hence, resource-restricted devices do not have enough computing power to run this middleware. The second type is the *cloud-based* IoT middleware. All the available functionality is in the cloud, exposed as APIs. The functionalities range from very simple to computation intensive tasks. Finally, the third type is an *actor-based* IoT middleware which is also a three-layered architecture. This middleware is designed to be light weight. Thus, it can be embedded in all layers no matter if it is the sensory layer or the cloud. Because of this light weight design, the computation units are distributed. For instance, an actor-based middleware is deployed on a smart watch, which does not include a storage service. This missing storage service can be added by downloading an actor from the cloud which provides storage access. The light weight architecture achieves the best scalability and latency. Furthermore, as seen in the example, the computational units can be extended by adding pluggable actors.

The main differences between these three middleware approaches lie in the support to new device types, the offer of different middleware services and the location where they can be embedded or deployed [40]. A disadvantage of the cloud-based middleware is the dependency to the providers. Firstly, they determine the functionality range and if they shut the service down, users do not have a middleware anymore. Additionally, you have to trust the providers that the stored data is not misused. In the service- and actor-based middleware it can be chosen how the data is used. According to security and privacy all middlewares provide solutions. Nevertheless, there exists a weak link between the physical objects in the service-based and cloud-based middleware because they can not be embedded in the physical devices.

2.3.4 Challenges

The IoT is currently in the phase of “Peak of Inflated Expectations” according to Gartner’s Hype Cycle [1]. It is expected that it reaches the “Plateau of Productivity” in two to five years. That means there exist still challenges which have to be solved. Hence, some of them are presented in this section.

One challenge is the security of information and protection of privacy [37]. The IoT devices, such as smart watches, make it possible to automatically collect personal data. Thus, a lot of private data is collected all the time. In addition, the infrastructure provides a lot more attack vectors than the traditional *information and communication technologies*. For example, when a company monitors the temperature of food in a cold store. In case the food deteriorates, the data is sent back to the company. By sending this data, it is critical to assert the privacy. In case of a leak it may harm the reputation of the company. Therefore, the protection of the privacy and the security of the data is an issue. Although existent network security technologies provide basic protection, there are still open topics. Thus, further research has to be done in fields like communication security or trust and reputation mechanisms.

Another issue is the heterogeneity of the network [36, 37]. Hence, improving the interoperability is a challenge. The cause for this problem is the lack of a platform which is widely accepted. This platform could hide the majority of the heterogeneity. Furthermore, the large amount of data, which is exchanged within the network causes delays, conflicts and communication issues. Thus, the challenge is to develop technologies and standards to transfer the data efficiently in the IoT network.

Regarding to services for customers, availability and reliability are key challenges [36]. Actually these two terms are related to each other. Reliability can be described as availability of information and services over time. Hence, availability means that IoT services are available anytime and anywhere. This is achieved by realizing it on the hardware and software side. In terms of hardware, the existence of a device must be guaranteed. This can be done by implementing redundancy in the system. Establishing software availability means to provide the services to everyone at different places at the same time. Also reliability has to be implemented in hardware and software. It aims to improve the service delivery and to meet the system requirements. However, the biggest bottleneck is the communication layer which has to be robust against failures. To realize a reliable IoT the communication, with regards to perception layer, data gathering, processing and transmissions, has to be reliable.

CHAPTER 3

Related Work

This chapter presents the related work which is relevant for this thesis. Based on the results of Chapter "Background", technologies, methods and approaches are introduced which are worth to consider for using them in the proof-of-concept implementation in this thesis. Furthermore, the state-of-the-art of use cases and existing approaches is explored. Therefore, Section 3.1 gives an overview about current solutions which are aiming to combine blockchain technologies and the IoT. Section 3.2 describes different blockchains and compares them. Because of the distributed environment of the IoT, communication is important, thus Section 3.3 presents different communication protocols. Finally, Section 3.4 gives an introduction about storing data in the blockchain and off-chain storage possibilities.

To follow the SLR approach which is presented in Chapter 2, the research questions for this chapter are as follows:

(RWQ1) Which different approaches exist to bring blockchain and the IoT together?

(RWQ2) What are the current issues?

(RWQ3) Which supportive technologies or principles also emerge?

(RWQ4) Which different blockchain approaches exist?

3.1 Blockchain and IoT

An alternative to its utilization as the ledger for cryptographic currency is the application of blockchains for challenges in the IoT environment. For instance, Dorri et al. [2] name privacy and security as features of the blockchain which are beneficial for the IoT. Nevertheless, literature about combining the blockchain and IoT is rare, according to the systematic literature review from 2016 of Conoscenti et al. [7]. This review identifies four use cases which are relevant for the IoT. These four use cases are: tamper-proof log of events and management of access control data [41], purchase of assets, such as sensor data, by devices or human beings [42], purchase of sensor data in IoT [6] and *public key infrastructure*s (PKIs) for management of updates, registration and revocation of keys [43, 44]. In the last two years, more and more papers were published, and the research community is active to achieve further progress. Besides the scientific community, there are also industrial efforts to take advantage of these new technologies. The following subsections will present some approaches in more detail.

3.1.1 Management of IoT Data

Huh et al. [45] introduce a proof-of-concept where policies for IoT devices are set on Ethereum via smart contracts. For the approach, they use a smart phone and three Raspberry Pis whereas the Raspberry Pis simulate a meter for electric usage, a light bulb and an air conditioner. The smart phone is used to set the policies for the connected devices. In the presented proof-of-concept the policy determines that the electricity consuming devices switch into a power-saving mode, when a certain threshold of energy-consumption is exceeded. To keep the air conditioner and light bulb up-to-date with the latest policy, these devices are regularly polling the smart contract for changes. Furthermore, the devices are also polling the meter contract. This contract is updated by the meter with the current energy consumption. In case the energy consumption is too high, the devices switch into a power-saving mode. As a result of this proof-of-concept, the approximate twelve seconds transaction time is stated as too slow for some domains. Also the lack of a light client for Ethereum is a problem for the restricted devices. In the meantime this problem is solved because Ethereum [46] released in November 2016 the first client application with a light client option. However, if this light client is suitable for the presented scenario has to be tested.

Liu et al. [47] introduce a blockchain-based data integrity framework for IoT data. This framework has four elements which are the *Data Owners Application* (DOA), *Data Consumer Applications* (DCAs), *Cloud Storage Service* (CSS) and the *Data Integrity Service* (DIS). Whereas, the DIS is implemented as a smart contract on the Ethereum blockchain. An illustration of the framework can be seen in Figure 3.1. Basically, the data is stored in form of data blocks on the CSS by the DOA. The integrity checking procedures slightly differ, depending on whether the DOA or the DCAs want to check the integrity. Furthermore, it is also considered if the CSS supports cryptographic functionalities. However, the basic procedure is to retrieve the hash of a data block from the CSS and from the DIS. In case these two hashes are equal, the integrity of the data is successfully checked. An advantage is that this framework is fully decentralized and therefore achieves a high efficiency and reliability of the DIS. One downside is that, when this approach was tested, it took 16 to 18 seconds until the blockchain reached a consensus and therefore, the hash of a data block is able to be checked.

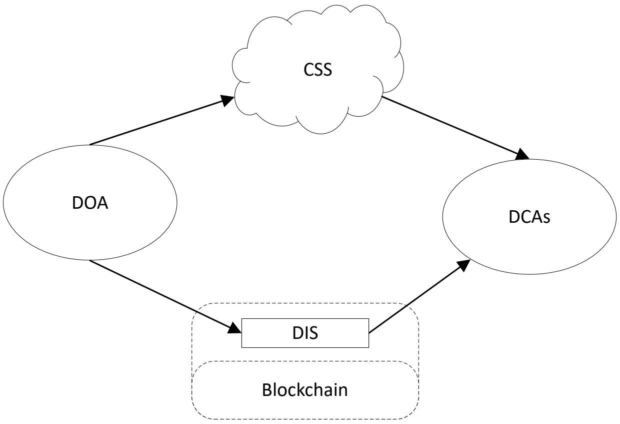


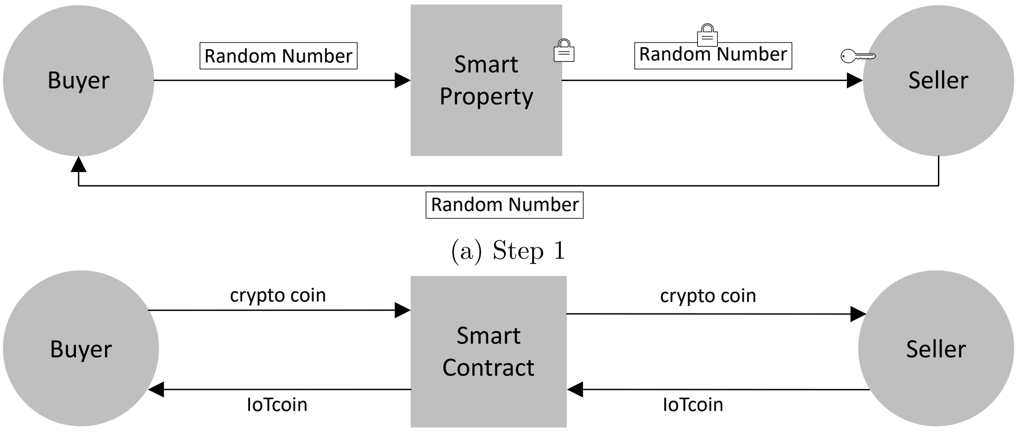
Figure 3.1: Data Integrity Service Framework (Source: [47])

Another approach [48] is the open-source software of the company Ubirch[1]. The aim is to log data tamper-proof, secure the data's origin and encrypt the data for the transmissions. The application collects data from sensors and puts the hashes of the data into the blockchain. Due to cost saving reasons, this procedure is done in two steps. In the first step, the hashed data is stored into the company's private blockchain. After that, hashes in the private blockchain are collected as bunches and added to the Bitcoin blockchain. Because of the bunches, more hashes can be added to a transaction. Bitcoin allows up to 40 bytes of data per transaction. With the bunches the 40 bytes can be used more efficiently. Hence, the transaction fee is split between all included hashes and therefore a client only has to pay a fraction of the fee for a data point. Besides the cost saving, this company focuses also on the security. To secure the data transmission and to determine the origin of the data, public-private encryption is used. However, to enable this security mechanism, every sensor has to own a private key. With this key, the data is encrypted and sent to the company where it is decrypted and stored as a hash in the private blockchain. Additionally, it can be verified which sensor has sent this data. Possible application fields are insurances. For instance, sensors can be built into cars and track the driving behavior. Depending on the driving habits, the payment for the insurance can be adjusted.

3.1.2 Business Approaches

Traditional E-business models do not suit the requirements which are needed for business models in the IoT world [49]. Since traditional E-business models need a third party, the usually applied P2P architecture of the IoT can not be used as an advantage. Zhang and Wen [42] propose an E-business model which uses the Bitcoin blockchain to overcome these issues. In their further development [49], they use smart contracts for the model and implement a prototype on Ethereum. The elementary components

[1] http://ubirch.de [Accessed: 2018-05-21]



(a) Step 1

(b) Step 2

Figure 3.2: Purchase of a Smart Property. (Source: [49])

of this model are *decentralized autonomous corporations* (DACs) which are transaction entities. These DACs can offer paid services completely autonomously. To achieve the independent operability, decisions have to be made without the involvement of humans. Therefore, decisions are made based on basic rules which are supported by machine learning techniques. The commodities of the new E-business are smart properties and sensor data. A smart property is a property which can be secured with an access control system or electronic lock. The trade of the commodities is not limited to the DACs only, also humans can be part of the E-business model. Hence, four combinations of trading relations are possible, namely seller and buyer are humans, both are DACs, the seller is a DAC and the buyer a human or vice versa. Figure 3.2 shows how the transaction of a smart property is handled. However, before this step, the buyer has to find the product of interest and compare different prices. The seller has to release the product, set a price and advertise the product. When buyer and seller have found each other, they have to negotiate and agree on the contract conditions. In case they have an agreement, the transaction can be issued. At first, the buyer challenges the seller to prove that he is the owner of the smart property. This is done by sending a random number to the property, where it is encrypted with the public key of the owner. In case the seller is the owner, he is able to decrypt the random number and sends it to the buyer. When the received and sent numbers are identical, the ownership is cleared and a smart contract is published into the blockchain. The buyer sends the required sum of coins to the smart contract and the seller sends an IoTcoin to the smart contract. Bitcoin is adapted as currency for the payment. The IoTcoin works as a commodity exchange certificate. It is used to exchange sensor data or smart properties. Hence, this coin indicates the ownership of the smart property. When the smart contract has received the money from the buyer and the IoTcoin from the seller, it disburses the coins to the corresponding parties. After the disbursement, the buyer is the owner of the smart property. Additionally, the smart property updates the ownership, when the block with the transaction, containing the

IoTcoin, has gained enough depth in the longest chain. After the update of the ownership, the purchase of the commodity is completed. By storing the ownership in the smart property like a car, it can be unlocked by the owner's private key. Thus, buyer and seller never have to meet for an exchange of physical keys. The whole purchase process is completely digital.

Another approach which focuses on the purchase of sensor data in Bitcoin is done by Wörner and Bomhard [6]. In this concept, a requester can directly purchase data from sensors in the Bitcoin blockchain. Three drawbacks for this solution are stated. First, the data sent in the transaction is visible to all. This can be solved rather easily by data encryption with the public key of the purchaser. The second problem deals with the validation mechanism of Bitcoin, where the approximate verification time of one block is ten minutes. That means it is not possible to deliver the data immediately and guarantee simultaneously no double spending. Finally, scaling is an issue. The blockchain will have lots of transactions which contain small sensor data. These transactions are stored forever on every node. Thus, each node has to provide a lot of resources.

Huckle et al. [50] propose to use the blockchain and IoT for shared economy applications like Airbnb[2]. Furthermore, these technologies can help to automate daily routines. Since the payment process is completely digital and it can be secured with smart contracts, the connection with other services is possible. For instance, a service could create a shopping list for groceries and send it automatically to the cheapest retailer. Another possibility is the search and payment for the closest parking space. The music industry is also a potential field of application. It can be used to make the value chain more transparent and to disburse the earnings of streaming service faster, especially to the artists. A company who has adopted the concept of using the blockchain and IoT in the shared economny is slock.it [51]. They are developing an universal sharing network which provides the infrastructure to deploy blockchain applications modules. This shall make it easier to add objects to the network and access them.

Concrete projects, which try to use the blockchain, arise for smart grids [52]. One of them is the "TransActive Grid" project in New York. The idea here is to manage efficiently rooftop photovoltaic installations. This is achieved by installing smart meters which report the current supply to the blockchain. Subsequently, also an automatic payment process shall be implemented.

3.1.3 Security & Privacy

As has already been explained in Section 2.3.4, privacy and security are important topics for the IoT. To use most of the security and privacy advantages of a blockchain for the IoT, Dorri et al. [2] present a lightweight blockchain-based architecture. This approach aims to eliminate the overheads of a blockchain, to improve the performance of the network. Therefore, a three-tier architecture is proposed. Basically, this blockchain-based architecture is designed for various use cases in the IoT. However, the approach

[2] https://www.airbnb.com [Accessed: 2018-03-28]

is explained on the example of a smart home. Thus, the first tier is the smart home environment. This environment comprises IoT devices, an immutable ledger and a local storage. The immutable ledger is similar to a blockchain, but it is managed by a smart home manager which controls all outgoing and incoming transactions. Furthermore, this private and local ledger contains policy headers which authorize received transactions. In case data of the smart home is requested by an authorized requester, the requested data is encrypted by the smart home manager with the public key of the requester. The second tier is a P2P overlay network. The nodes in the network can be the smart home managers, smart phones or personal computers. The nodes are structured in clusters, to reduce delays and the network overhead. Every cluster has a cluster head which maintains a list of public keys where it is defined which nodes are allowed to access data and which nodes are allowed to be accessed. All the cluster heads maintain a public blockchain where every node's transaction history is stored in the ledger. For performance reasons, the validity of new blocks is checked with the concept of distributed trust. Every cluster head stores a trust rating table of the other cluster heads. When a new block is generated, the block and a multisig transaction is sent to neighbor cluster heads. The neighbors check the trust rating of the issuing head or the cluster heads who signed the multisig transaction. The higher the trust the less random transactions in the block have to be validated. This decreases the validation time of a block. In case a block is generated by more than one cluster head, the cluster head's block with the highest trust rating is selected. The third tier is the cloud storage. Besides the local storage, data can also be stored in the cloud. Therefore, the data is stored in blocks with a unique block number. For authentication, the smart home manager uses the number and the hash. Simulation results with 50 nodes running for one minute show that traffic overhead is reduced by approximately 73% in comparison to Bitcoin. Furthermore, the processing time with distributed trust is roughly 50% of the processing time without the distributed trust concept.

Another approach to increase privacy for personal data is the Enigma platform [41]. This decentralized computing platform is not explicitly designed for the IoT, but the IoT is a potential use case. Since privacy on the blockchain is a problem, the idea is to store the data off-chain in a *distributed hash table* (DHT). Furthermore, a blockchain is not suitable for heavy computations, hence, private and heavy computations are executed on an off-chain network. To ensure the privacy, the computations are executed on the raw data without having access to it. For instance, if someone wants to know the average duration of study at a university, it is not necessary to know the duration of study for every single student. To realize such applications, the concept of private contracts is introduced. Basically, they are smart contracts with additional features. One difference is the keyword "private". The keyword allows developers to define private objects which means only the reference not the data itself is available. For the execution of the contract, an interpreter splits the code. The public part of the code is executed on the blockchain whereas the private part runs on the Enigma network. This procedure improves the run-time while maintaining privacy and verifiability. The approach is continuously developed in the Enigma project [53]. One goal for 2018 is to release a private contract engine.

Furthermore, the private contracts shall be compliant with the Ethereum blockchain. The aim is to have no difference between developing a smart contract or a private contract, except of the syntax for private executions.

3.1.4 Discussion

Based on the results of this section, it can be said that the blockchain has several advantages which are of use in the IoT domain. Additionaly, there has been further research in this field since the SLR of Conoscenti et al. [7]. Nevertheless, there are still many issues and challenges to be solved for more complex use cases and the improvement of the software development process in this domain.

Low Level Approaches

A lot of the papers deal with the topic blockchain and IoT on a low level. It means that the research is focused on basic questions of the blockchain and IoT. Thus, there is little about integrating this technology in more complex applications. The challenges to make a blockchain more suitable for IoT, may be a reason for the focus on the basic questions. However, companies like slock.it or ubrich are currently working on integrating these technologies in sophisticated software products. According to the science side, research on exception handling and best practices would be beneficial for more complex applications and the development on a higher abstraction level.

Hybrid Approaches

As has already been stated, the blockchain has some problems to overcome to be used in the IoT. For instance, privacy and block generation times are some of the challenges. Our research results show that hybrid architectures like off-chain storage or a combination of private and public blockchain is a common solution. The challenge with this hybrid approaches is the existing trade-off. On the one hand, blockchain problems can be solved by off-loading certain operations, but on the other hand, the risk to interfere the system too much and therefore lose advantages of the blockchain is omnipresent. The combination of a private and public combination is a good example for the trade-off. The private blockchain has a better performance but the data is not visible to the public. Hence, some of the transparency of a public blockchain is lost. Therefore, the trade-off has to be considered for the design of hybrid-based architectures.

Little Diversity

Besides a self developed blockchain-based architecture, the only used chains are Bitcoin or Ethereum. One cause for this may be the existence of many blockchains which do not have a reliable development community in the background. Nevertheless, it would be interesting for a deeper understanding of the domain if there is a greater diversity on blockchains which are used for research. At least, if the design decisions for a specific blockchain are explained in more detail.

3.2 Blockchains

According to CoinMarketCap [54], there exist 1591 (March 2018) cryptocurrencies. From these 1591 cryptocurrencies, 916 can operate independently. The others rely on an already existing cryptocurrency. Despite the huge offering, Nordrum [55] says that blockchains do not really offer a benefit in comparison to current database or message solutions, especially in the financial industry. An issue is the lack of an existing standard or rather a widely accepted architecture of a blockchain network. Because of the miscellaneous tools and techniques, it is quite hard to compare the different blockchain approaches. Nevertheless, this section introduces three blockchains which give the impression to suit as a blockchain in the IoT. The criteria for the selection are the defined objectives, the maturity of the development community and the support by well-known companies. As a result Ethereum[3], Hyperledger Fabric[4] and IOTA[5] are presented in this section.

3.2.1 Ethereum

The Ethereum blockchain follows the same paradigm as Bitcoin but in a more general way [19]. A central technology for the universality of Ethereum is a Turing-complete programming language. It enables developers to build their own decentralized applications which are called smart contracts [56]. As it has already been explained in Section 2.2.2, the language is only theoretically Turing-complete because of a limiting parameter. This parameter is part of the fee model in Ethereum where every programmable computation, like contract creation or executing an operation on the EVM, creates costs. This shall protect the network against abuses. Another technology for the security of the system is the own algorithm for the PoW called *Ethash*. Ethash is based on the Dagger Hashimoto algorithm which aims to be ASIC-resistant and for light clients easily verifiable [57]. The objective of the PoW is to find a nonce input for the algorithm where the output is smaller than a certain threshold [32]. To enable a block generation approximately every twelve seconds, the difficulty to find a certain hash is adjusted over time. For the calculation of the hash, subsets of a data block are chosen dependent on the nonce and the block header of the current block. The data is newly generated every 30,000 blocks. Therefore, Ethash is memory hard and ASICs can not fully demonstrate their strengths.

The fast block generation is made possible by the *Greedy Heaviest Observed Subtree* (GHOST) protocol [56]. GHOST is needed because fast block generations cause security risks because of high stale rates. Since every block needs a certain amount of time to be propagated through the network, it happens that, for instance, miner A mines a block and miner B also mines another block before the block of A is propagated to B. In this case the miner B's block is staled and does not contribute to the security of the network because it is not used. The second problem which results from staled blocks, is centralization. Miners with high hashpower are more likely to produce less

[3] https://www.ethereum.org [Accessed: 2018-04-04]

[4] https://www.hyperledger.org/projects/fabric [Accessed: 2018-04-04]

[5] https://iota.org [Accessed: 2018-04-04]

stale blocks. Thus, miners will try to join the largest mining pool. A mining pool is a group of miners which combine their hashing power for the block generation to increase the chance of finding the first valid nonce [12]. Every member searches a different part of the nonce search space. In case of a successful block generation, the reward for the block is shared within the mining pool. Joining the largest mining pool will result very likely to a mining pool which owns enough percentage of the network to control it [56]. The GHOST protocol solves the security issue by considering the stale blocks for the calculation of the main chain. The staled blocks which are not part of the main chain but considered for the calculation are called *uncles*. The centralization problem is solved by paying also a reward to the stale blocks.

3.2.2 Hyperledger Fabric

Before starting with the introduction of the Hyperledger Fabric blockchain, the terms *permission-less* and *permissioned* ledger must be explained. Permission-less ledgers do not have a restriction on users joining the network [15]. In contrast, permissioned ledgers demand for permissions of authorities to join the network or trigger a transaction. The granularity of the restrictions depends on the actual blockchain.

Due to the good manageability of permissioned blockchains, the financial industry is highly interested in this kind of ledgers [55]. Banks, credit card providers and tech companies form consortiums to explore the potential of the blockchain. One of the consortiums is the Hyperledger consortium where Cisco, IBM, Intel, American Express and J.P. Morgan are members. A part of Hyperledger is the Hyperledger Fabric project [58]. This project is open-source and supervised by the Linux Foundation[6]. A special feature of this blockchain is the modular and extensible architecture which aims to be flexible, scalable, resilient and confidential. Furthermore, the concept of smart contracts is also supported, however, it is different from other approaches. Other smart contracts, for example in Ethereum, follow an *order-execute* architecture which has many limitations. One is the sequential execution of the transactions. Since no parallelisation is possible, the performance is suffering [58]. Another issue, in terms of confidentiality, is the execution of the contract on all peers. To solve these problems, Hyperledger Fabric introduces the *execute-order-validate* architecture which enables parallel execution and the execution on a subset of peers. A feature which improves the development process is the support of standard programming languages.

The success of this modular permissioned distributed ledger is reflected on 400 different prototypes, proof-of-concepts and also production systems, which are using Hyperledger Fabric [58].

3.2.3 IOTA

IOTA aims to be a cryptocurrency for the IoT industry [59, 60]. Therefore, it enables M2M micropayments by not claiming fees on the transactions in the network. Furthermore,

[6] https://www.linuxfoundation.org [Accessed: 2018-04-16]

IOTA is designed to scale well and to operate in an environment with lossy connections. In contrast to blockchains, like Bitcoin, the underlying structure is a *directed acyclic graph* (DAG) which is called the *tangle*. In the graph, the vertices symbolize the transactions and the edges are their approvals. When a node wants to issue a transaction, it has to approve two other transactions which are not approved yet. In terms of the graph, this means that the new vertex has to draw two edges to two different vertices which have no incoming edges yet. The name of non-approved transactions is *tip*. Normally, there exist more than two tips in the graph, therefore a special algorithm selects two vertices. After that, a PoW has to be done, to protect the network against spam and Sybil attacks. However, the puzzle is not as hard as in the Bitcoin network.

For achieving a consensus in the network, the method of the tip selection is very important [61]. Basically, the selection algorithm is a biased random walk [62] which starts at the genesis block, the first block of the graph. This type of algorithm is called *Markov Chain Monte Carlo*. From the genesis block, the algorithm searches a way through the graph to a tip vertex, by following the incoming edges in reverse order. In case the current vertex has more than one incoming edge, the edge labeled with the highest weight is more likely to be chosen. To avoid that tips are left behind, the bias of the weight on the selection probability can be adjusted. After the tip selection and establishment of approval edges, the *confirmation confidence* can be calculated which is an important value for the consensus term in the network. This confirmation confidence describes the percentage of tips which are approving a transaction. When the percentage passes a certain threshold, for example 95%, it is very likely that this transaction stays in the consensus. However, currently too few transactions are issued in IOTA to prevent double-spending attacks properly. Therefore, a voluntary and temporary consensus mechanism, the coordinator, is used [61]. The consensus works by issuing a *milestone* transaction every two minutes. The transactions which are approved by the milestone transaction have immediately a confirmation confidence of 100 percent. Issuer of the milestone transaction is the IOTA Foundation. As soon as there are enough transactions issued, the coordinator mechanism will be shut down [61].

An additional feature of IOTA is the *Masked Authenticated Messaging* [63]. It is a module that provides access to encrypted data streams in the network. The advantage of this service is that the integrity and tamper-proofness of the data can be guaranteed.

3.2.4 Comparison

The previous sections show that the projects are similar in terms of the basic idea of a blockchain but the concrete aims and approaches are different. Table 3.1 summarizes some facts about the presented projects. To categorize the blockchains, two types, namely *permission-less* (PermL) and *permissioned* (Perm), are used. Furthermore, to get an impression about the activity of the communities, the amount of commits in the official repositories are documented. The rate is calculated by dividing the commits by the amount of months, since the project has started. This makes the activity more comparable. Another calculation has been done for Ethereum's tps. For that, the maximum amount of

Table 3.1: Summary of the Blockchains' Basic Facts.

Blockchain	**Performance**		**Type**	**Smart Contract**	**Community**	
	tps	Latency			Commits	Rate
Bitcoin [12, 15, 56, 64]	4-10	10 min	PermL	~	16883	164
Ethereum [15, 19, 32, 65, 66]	16	12 sec	PermL	✓	9541	187
Fabric [58, 67]	>3500	<1 sec	Perm	✓	5894	256
IOTA [60, 61, 68]	>100	<10 sec	PermL	-	1473	87

transactions which has been recorded, 1,349,890 (4^{th} Jan. 2018) [65], is taken and divided by the amount of seconds of one day (86,400). Bitcoin is added to have a reference value. The smart contract status of Bitcoin is only marked with a tilde because it does support scripting. Scripting can only be seen as a weak version of a smart contract [56]. One reason for that is the lack of Turing-completeness. Also the scripts are not able to read blockchain data like nonces, timestamps or hashes of previous blocks.

At first glance, it is obvious that all blockchains achieve a better performance than Bitcoin. The best performance has the Hyperledger Fabric blockchain. Additionally, the commit rate is also the highest. This high rate in combination with the Linux Foundation as supervisor [58] suggests a strong development community in the background. Depending on the use case, the permissioned property can be an issue because of the existing authorities in the network. Hence, it is not fully decentralized anymore. Therefore, the risk of a single point of failure is higher [15] than on permissionless chains. Also Dinh et al. [69] say that Hyperledger outperforms Ethereum but if it is scaled up to more than 16 nodes it fails. Additionally Ethereum is more resilient to node failures.

According to the performance, Ethereum is way better than Bitcoin and in terms of latency only a little bit slower than IOTA. Considering the community, it has the second highest commit rate which indicates that it has an active development community. An advantage is the support of smart contracts with a Turing-complete language.

IOTA achieves the best performance of the permission-less blockchains. However, the lack of a smart contract support is a disadvantage. Another downside is the low commit rate. Although this does not have to be a bad sign, but the need of a coordinator, to achieve consensus, is another indicator for little activity in the community.

To sum up, it can be said that IOTA seems currently not mature enough to be used in projects. Hyperledger Fabric is interesting in terms of the performance, however, the use of a permissioned ledger must suit the use case. Ethereum seems to be a further development of Bitcoin. Therefore, it has the advantages of a fully decentralized blockchain with a better performance and smart contracts.

3.3 Communication Protocols

Communication plays a key role in the IoT, according to Talaminos-Barroso et al. [70]. Therefore, the decision of the right communication protocol is crucial to develop a performant application. This section introduces the *Constrained Application Protocol* (CoAP), *Message Queuing Telemetry Transport* (MQTT) and *Data Distribution Service* (DDS) and discusses the advantages and disadvantages.

CoAP is a protocol based on *Representational State Transfer* (REST) [36]. However, it is optimized to suit the requirements of an IoT environment. Thus, the protocol has a low power consumption and is able to operate despite lossy and noisy links. It supports two communication mechanisms, namely request/response and publish/subscribe. The observing functionality of the publish/subscribe mechanism is ideal to monitor resources, for instance sensors. Another specific functionality is the resource discovery. Therefore, a client is able to request the available resources of the server. As a response, the client receives different *uniform resource identifier*s (URIs) which can be requested.

MQTT is a protocol which works with the publish-subscribe pattern [36]. Like CoAP, it is also optimized to operate on resource-restricted devices with lossy and low bandwidth links. To enable this messaging protocol, three components are needed namely a publisher, subscriber and broker. The publisher generates data and sends it to the broker where it is forwarded to the subscribers. To receive only data of interest, subscribers assign to certain topics. Hence, they get only data which is published to the assigned topics. Furthermore, this protocol supports three levels of *Quality of Service* (QoS). The first level states that a message is delivered at most once, no further acknowledge is required [71]. The next option is to send the message at least once. It is not guaranteed that there does not exist a duplicate of the message. This is solved on the last level.

DDS can be described as a publish-subscribe protocol [72]. The special feature about it is, that it does not need a broker, like other publish-subscribe protocols. Furthermore, DDS targets real-time systems and has an extensive support of QoS. For the basic communication, five entities are needed. The *Publisher* is responsible for the dissemination of the data whereas the *DataWriter* is the facade of the Publisher. The DataWriter is used to communicate data values and changes. The *Subscriber* receives the data which is accessible via the *DataReader*. A DataWriter and a DataReader are associated by a *Topic*. The Topic defines a unique name, a data type and a QoS definition.

A general ranking of the protocols can not be made because the performance often depends on the conditions. MQTT has a lower delay than CoAP when the package loss is small [73]. In case the package loss increases, CoAP achieves better results. A strength of both is the good retransmission scheme. The DDS protocol fits for real-time applications [70]. Furthermore, it supports automatic discovery, has a sophisticated QoS and scales well. The downside is the high memory consumption and regarding development open source libraries are rare. In contrast, MQTT has an active developer community and is lightweight in terms of memory and CPU consumption. CoAP is lightweight as well but it has, like DDS, a deficit on libraries and tools. To sum up, it can be said that DDS is perfect when real-time is crucial and a sophisticated support of QoS is required. With a low sampling rate, MQTT also suits real-time applications but it is a simpler implementation of the publish-subscribe pattern. Regarding the development, CoAP has some deficits. However, the resource discovery, the lightweight design and multicast functionality make it a useful protocol for the IoT.

3.4 Storage

Xu et al. [15] state that it is a common technique to store data off-chain. The payload which is stored on-chain is kept small, like meta data or hash values of the raw data. In Bitcoin, this approach is useful because the payload size for transactions is limited. Besides the limitation it saves money. The reason for that is a fee charged for every byte by Bitcoin and also Ethereum. For instance, 80 Bytes roughly cost 0.47$[7] in Ethereum. When the data is stored in a 32 byte variable of a smart contract, it is even more expensive. For the 32 bytes, 0.76$[7] have to be paid. A cheaper way to store the data into the smart contract is the log event option. Choosing this option 32 bytes cost 0.38$[7]. In comparison, Bitcoin charges 0.56$[7] for a transaction with 80 bytes payload. Therefore, having a technique which reduces data stored on the blockchain saves money and overcomes storage limitations. Besides storing data in the cloud, P2P storage is a possibility. The *InterPlanetary File System* (IPFS) is such a storage which suits blockchains.

The IPFS [74] is a P2P distributed file system. Among other existing technologies it combines concepts of DHT, the filesharing system BitTorrent[8] and the version control system Git[9]. The aim of this project is to connect all computing devices with one common file system. An advantage of this architecture is that it has no single point of failure. Furthermore, the nodes in the network do not have to trust each other and no one is privileged. This seems similar to the concept of permission-less blockchains. The unique identification of the content is done by its *multihash*. The multihash is a specific format which adds meta-information to the hash of the content. Hence, the multihash comprises a function code which specifies the hash function, the digest length and the digest bytes. Ali et al. [75] use this storage for a decentralized access model for IoT data. Therefore, data is stored in IPFS and the hashes of the files are persisted in the blockchain.

[7] Same calculation as in [15] only the current prices (US$ 9349.56/BTC, US$ 703.75/Ether) are taken from https://coinmarketcap.com [Accessed: 2018-04-24]

[8] http://www.bittorrent.com [Accessed: 2018-04-24]

[9] https://git-scm.com [Accessed: 2018-04-24]

CHAPTER 4

Solution Approach

One goal of this thesis is to evaluate the performance of different IoT devices. Therefore, an application is developed which combines blockchain and IoT technologies. This application represents a prototype of an application which can be adapted for real world scenarios. Thus, the design of the application in this thesis combines, on the one hand, the part which can be used for real-world use cases, on the other hand, the performance evaluation is also considered. How this is realized in detail, is part of this and the following chapter.

Section 4.1 summarizes the benefits and challenges of combing a blockchain and the IoT. This forms the basis for the formulation of the concrete research questions. Consequently, these research questions shall be answered by the implementation of the application and the evaluation of the performance on the different IoT devices. The following sections go into more detail about the design of the application. Section 4.2 analyzes the requirements for the application and the performance evaluation. In the next Section 4.3, the architecture of the application and the integration of the evaluation part is discussed. Finally, Section 4.4 argues about the decisions for the used technologies.

4.1 Research Challenges

The blockchain provides promising opportunities for the IoT [76]. The public key cryptography and hash algorithms can be used in the IoT to hide the identity of a participant in the network. Another opportunity is the monetary exchange of data and computing power. For instance, in smart cities these services can be provided with the engagement of the community. To incite the community, rewards are paid for the involvement. A further opportunity is the record of transactions on the blockchain for account and audit. The blockchain is decentralized, hence the infrastructure is owned by several organizations. This decentralization is beneficial for supply chain monitoring, since traditional supply chain monitoring is done on a centralized architecture [76]. Storing the data in a decentralized ledger increases the trust. An additional opportunity for the IoT are smart contracts. Smart contracts can be used as an alternative to paper-based contracts. The reason for that is the possibility to exchange assets with non-trusted parties whereas the smart contract ensures the validity of the deal.

Kshetri [77] also says that the blockchain is able to address some challenges of the IoT in comparison to the cloud model. The cloud model comprises cloud servers which identify, authenticate and connect IoT devices. Furthermore, storage and processing can be done on the servers. The blockchain provides potential solutions to some challenges of the IoT. One challenge is exponential growth of IoT devices. The costs to enable the communication between this huge amount of devices will also increase exponentially when it is realized with a centralized cloud model. Thus, the blockchain with its decentralized architecture can be beneficial. The communication between the IoT devices can be realized with smart contracts. Another disadvantage of the centralized cloud model is the occasional downtime due to various reasons. The blockchain does not have this problem, since there is no single point of failure in the blockchain's P2P architecture. Furthermore, the blockchain adds security to the message exchange between devices by signing the transactions. Hence, the originators of transactions can be verified.

Besides the opportunities the blockchain may provide to the IoT, there are also challenges when the blockchain is applied in the IoT [76]. IoT platforms have restricted resources in terms of computation, communication and storage. However, the blockchain technologies need a lot of resources. There exist light clients, like in Ethereum [78], which try to reduce the overloads of the blockchain synchronization and consequently the resource utilization. Nevertheless, the question remains if these light clients are able to overcome or rather ease the resource constraints. Latency demands are another challenge [76]. IoT applications can be described as a collection of data producers and consumers. Sometimes, data consumers want to react to received events. This behavior is less responsive, if blockchain technologies are used because of the long block times of contemporary blockchains. It can be said that time-sensitive IoT applications which depend on fully confirmed transactions, are not suitable with the current blockchain technologies [76]. Another issue is, that in case sensors or actuators get hacked or used in a wrong way, the data is then logged in the blockchain and can not be changed anymore. Furthermore, the intermittent connection of devices is a challenge. Not every device is always connected, for instance, devices which are used for supply chain monitoring or devices which run on battery and save energy, have intermittent connections. To keep itself synchronized, it has to synchronize the blockchain when the connection is established. However, the question is if this is worth all the costs in terms of computation, bandwidth and storage. Distributed ledgers which follow a DAG-based approach like IOTA [60] provide mechanisms to make transactions in an offline environment and merge the partitions later on with the network in a less costly way [76].

In the conducted research process during the course of this thesis, further challenges are identified which are presented in Section 3.1.4. One noticeable result is that only a small amount of papers deal with blockchain and IoT integration on a higher abstraction level. Another discovery is that the diversity of blockchains used for the research is not high. Based on the research results of Chapter 2, 3 and this section following research questions are formulated for this thesis:

RQ-1 How can the integration of the blockchain in the IoT be realized from a software engineering perspective?

This research question aims to show on a concrete example, how such an application can be designed. Therefore, a middleware-based approach is used. Thus, the application is not one centralized program, but it comprises several parts. The first part deals with the collection of data from sensors and transferring data to the middleware. In the second part, a middleware has to be designed which is able to distribute the data via different channels.

RQ-2 What is needed to suit time-sensitive applications and take advantage of blockchain properties?

This question aims to deal with the problem that current blockchains are not suitable for time-sensitive applications [76]. Hence, a solution is developed which, on the one hand, enables a time-sensitive handling of the data and, on the other hand, takes advantage of blockchain properties.

RQ-3 How to store data while keeping the stored data volume in the blockchain low?

The final question deals with the storage of the sensor data. As it has been said in the previous chapters, especially in Section 3.4, unlimited data storage in the blockchain is either not possible or it is expensive. Furthermore, it is a question of scaling [6]. Therefore, this question aims to minimize the data which is stored in the blockchain, whereas the P2P approach is maintained. The P2P approach avoids a centralization of the system.

RQ-4 How does a blockchain-IoT application perform on different devices?

Since the IoT is an environment with heterogeneous devices, it is interesting to explore how these different devices perform. Therefore, this question aims to test the developed application on three different devices. The results shall show if and how the performance of the application is affected by the executing device.

4.2 Requirements Specification

This section defines the requirements for the blockchain-IoT application of this thesis. The aim of this requirements specification is to lay a foundation for a middleware which can be used in general for IoT devices and applications which want to integrate the blockchain. Since the application is used to find answers to the research questions, the requirements are derived from the questions and the research results. Furthermore,

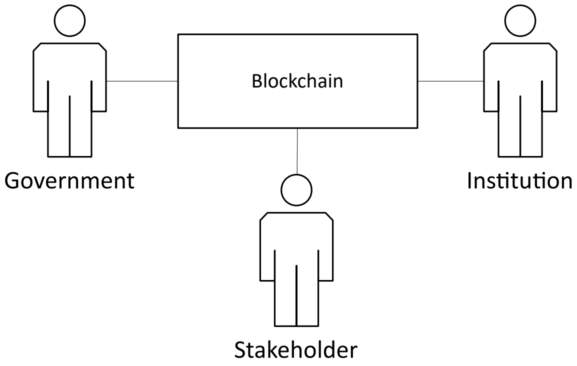


Figure 4.1: Stakeholders of Motivational Scenario.

the regulation of the European Union [4] for monitoring and reporting greenhouse gas emission is used as motivational scenario to specify the requirements.

4.2.1 Motivational Scenario

The regulation on a mechanism for monitoring and reporting greenhouse gas emissions of the European Union is a framework which is established to evaluate the progress in achieving greenhouse gas emission goals [4]. Member States must send annually a report to the European Commission with an analysis about the current status of greenhouse gas emissions. To achieve a proper evaluation of the progress, the quality, immutability and consistency of data is important. Thus, the application of the blockchain in this scenario could increase the transparency and guarantee immutability of the data. The idea is that environmental data can be collected by an IoT sensor network which stores the data directly into the blockchain. Primarily, this would guarantee tamper protection because once a transaction is added to the blockchain, it can not be changed any more. Another advantage is that the measured values would be accessible immediately and the European Commission and other stakeholders could monitor data in nearly real-time without waiting for the annual report. Figure 4.1 shows the stakeholders for this scenario. The government feeds the blockchain with sensor values whereas the institution, like the European Commission, retrieves the values and uses them for further processing. The stakeholder can have different roles. For instance, he can access the public data to establish own services or he installs an own control instance. Based on this idea and the introduced stakeholders, specific use cases and requirements are derived in the following sections.

4.2.2 Use Cases

In general, use cases describe the interaction between a system and an external actor [79]. The actor itself is not necessarily a person, but it can also be a software system or a hardware device. The use-case approach focuses on the tasks of the actors. The aim is to find out what the actors need to interact with the system.

To get a broad analysis of the system requirements, use cases are specified for the application of this thesis. It helps to define the functional and non-functional requirements in the following sections. The use cases are identified with regard to the opportunities of the blockchain, the properties of the IoT and the motivational scenario of the European Union. Thus, the use cases are placed in a governmental context or at least in an environment of big organizations. However, the solution is not limited to these use cases. The use cases support the requirements analysis and help to get a better understanding of the usability and potential application areas.

Use Case 1: Access Data in Real-Time

- **Scenario:** The website of a weather station wants to publish the current temperature of a specific region. Therefore, it has to know the endpoint from where it wants to receive the data. If this is the case, the web server connects to the endpoint and subscribes to a real-time data stream. After that, the web server listens to incoming messages and publishes the latest temperature data on the website.
- **Actor:** Can be anyone or anything which is interested in some public data, for instance a web server of a weather station.
- **Precondition:** The actor knows the endpoint from where it wants to receive the data.
- **Postcondition:** The actor has subscribed to the real-time data stream and receives data.

Use Case 2: Access Data with Guaranteed Integrity

- **Scenario:** State authorities want to monitor the greenhouse gas emissions and write an annual report about the current state. Therefore, the integrity of the data shall be guaranteed. Hence, state authorities subscribe to a communication channel where the integrity of the data can be guaranteed.
- **Actor:** State authorities/ Institution
- **Precondition:** The actor knows the endpoint where the data can be received and which data is desired.
- **Postcondition:** The actor receives data for which the integrity can be guaranteed.

Use Case 3: Push Continuous Data Delivery

- **Scenario:** For monitoring, missing values are not desired. Furthermore, if the monitoring party (data consumer) is used to control the data producer, the data producer may have the motivation to miss sending values. This can happen to hide the exceeding of certain thresholds. Thus, the data consumer wants to ensure that the data is delivered by demanding penalties for missing values. An agreement is made between data producer and consumer. The agreement is controlled automatically by an independent instance like a smart contract to avoid tampering. In case of a breach of the agreement the penalty is executed also automatically.

- **Actor:** Data consumer, Data Producer
- **Precondition:** The data producer and consumer have made an agreement. In case of missing to deliver a value within a specified period, the data producer has to pay a penalty.
- **Postcondition:** The penalty is paid.

4.2.3 Functional Requirements

Functional requirements describe which functionality shall be supported by the software [79]. This functionality enables the user to fulfill its tasks.

For the application of this thesis, the following functional requirements are specified:

1. **Collection of Sensor Data**

   The application shall be able to collect data from sensors. Furthermore, the collected data shall be provided via an interface.

2. **Access of Data in Real-Time**

   The sensor data shall be accessible in real-time. For this data access type, the integrity has less priority.

3. **Access of Data with Guaranteed Integrity**

   The data which is collected shall be accessible in a way where the integrity can be ensured. For this data access type, the time-sensitivity has less priority. The important thing is to guarantee the integrity.

4. **Automatic Check on Update Frequency**

   The update rate of sensor data shall be checked automatically.

5. **Penalty Payments**

   In case sensor data is not delivered in a defined period, a penalty is automatically paid from the data producer to the data consumer.

4.2.4 Nonfunctional Requirements

Besides the functional requirements, there are also nonfunctional requirements [79]. These nonfunctional requirements can determine performance goals of described quality attributes. Quality attributes extend the specification of a product's functionality with its characteristic. These characteristics can be specified not only for the requirements of the user but also for the developer's needs.

For the application of this thesis, the following nonfunctional requirements are specified:

1. **Usage of IoT Standards**

   Although, the heterogeneity of the IoT network and the lack of a standard is mentioned in the literature, there are still technologies which have proven their value. The usage of IoT standards shall help to fit the application into the IoT environment. Furthermore, standard technologies implement already needed features for the IoT, for instance the low power consumption of CoAP [36].

2. **Extensibility**

   Extensibility is important because of the middleware-based approach. The middleware shall be extensible to add more communication technologies and to be adaptable in general for the heterogeneous IoT environment.

3. **Portability**

   The application shall not be limited to only one platform. Therefore, the portability has to be considered during the development process. This requirement suits well into the heterogeneous environment of the IoT.

4. **Save Resources**

   The application shall consider the resource restrictions of IoT devices. These requirements may be in conflict with other requirements. In case the application would loose functionality, saving resources shall be prioritized less. As it has been said, this nonfunctional requirement is already partly achieved by using specific IoT technologies.

4.3 Architecture

In this section, the design of the application is presented, including the part which is used to measure the performance metrics. The architecture is designed to fulfill the specified function and nonfunctional requirements.

Figure 4.2 shows the design of the solution for this thesis. The drivers, sensor and virtual driver, the middleware with its both channels form the blockchain-IoT application. Thus, the specification of the requirements belongs to those elements. The IoT client aims to handle the performance evaluation which is needed to test the application on different devices. Different performance metrics like delays are observed on the IoT client. Additionally, the activity of the blockchain is monitored. Thus, the IoT client does not suit real-world scenarios, but it shows that the interfaces of the middleware work. Hence, the client can be exchanged with another client which fulfills requirements which are needed for real-world scenarios. The different elements of the solution are presented in the following subsections, beginning with the sensor drivers.

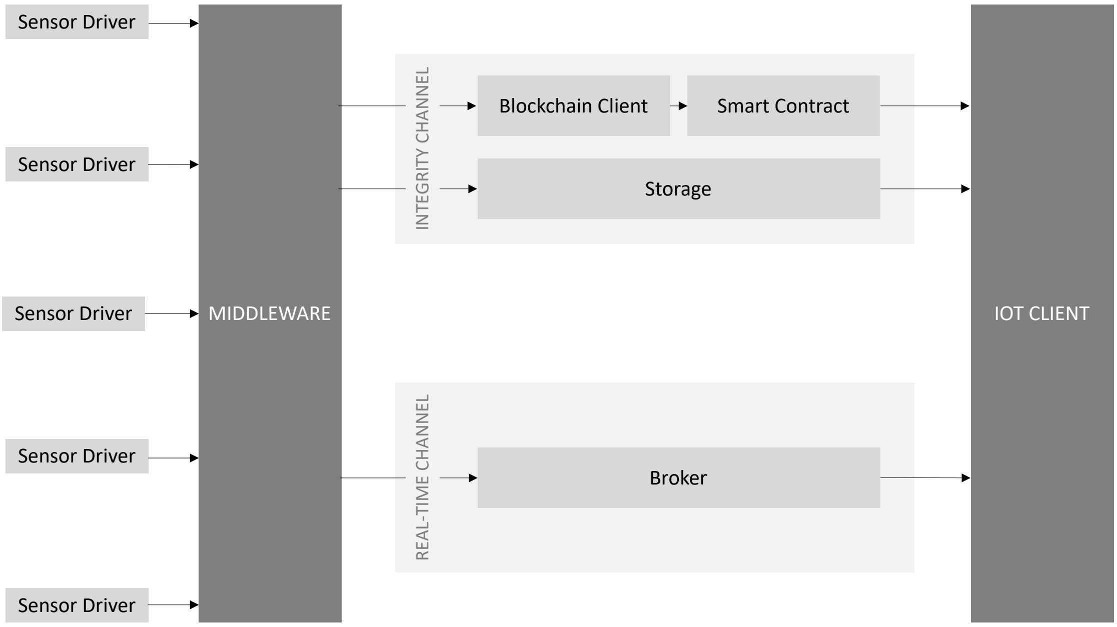


Figure 4.2: Architecture of the Solution Approach.

4.3.1 Sensor Driver

At first the terminology for this element has to be cleared. A *sensor* is the hardware which does the sensing of environmental phenomena [80]. A *phenomenon* is the entity of interest which is sensed like temperature or humidity. Additionally, another term is added in this solution approach. Every phenomenon is managed in the driver by a *data source* which collects the values of one phenomenon from the sensor. Furthermore, the data source packs the measured values into messages and adds meta-data like timestamp and type of measurement.

The sensor driver acts as an interface between the sensor and the middleware. Thus, it is responsible to collect the data from a specific sensor. A sensor driver only implements the collection mechanism for one sensor. That means every sensor has its own driver. However, it may be the case that one sensor senses several phenomena. In such a case, the sensor driver collects the values for all phenomena which the sensor supports. Furthermore, the collected data must be provided via an interface to the middleware. Therefore, the driver provides the phenomenon's data via IoT-suitable communication channels. Every phenomenon has its own channel. However, before the exchange between the driver and the middleware can be realized, the driver has to register at the middleware. The idea behind this process is that sensors can be added to the middleware dynamically. This avoids a hard coding of driver connection into the middleware and therefore adds flexibility. The workflow of the registration procedure is constituted in Figure 4.3. At first, the driver sends a registry request to the middleware containing its own address. This address is needed by middleware to know where it has to send the answer. When the registry

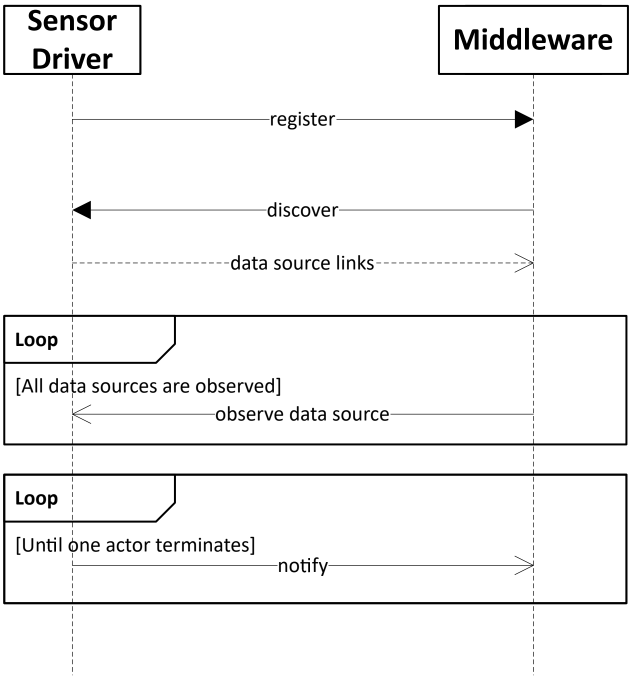


Figure 4.3: Sensor Driver Registry.

request is received, the middleware responds with a discovery request. This request wants to get the links for the different data sources of the driver. The driver responds with a list of data source links. For every resource link, the middleware sends an observe request. That means in case the observation is accepted by the driver, the middleware is notified, if a new value is received. Depending on the driver's implementation, there could also be another reason. For instance, the middleware is only notified when the sensed value has changed. Although, the operation is called notify, the middleware is not only notified but the new value is already included in the notify message. As long as the middleware or the driver is running, sensor data is delivered to the middleware.

Besides the dynamic adding of sensors to the middleware, the separation of the sensor driver from the middleware has another advantage. Because of this strategy, the implementation of the driver is independent from the technology of the middleware. Thus, an own programming language can be chosen to collect the data from the sensors in an optimal way. This can be an advantage because different programming languages aim at different application fields [81]. Since the driver and the middleware fulfill different tasks, the independence makes sense. The only constraint for the driver is to support the interface which enables the workflow of Figure 4.3.

Since the application also aims to execute performance tests, there are two types of sensor drivers. The first type implements the collection of sensor data as it has been explained before. However, the second driver does not have a real connection to a physical sensor. Thus, the driver operates only as a virtual driver. The virtual driver is able to start a specified amount of data sources. The data sources create messages with hard-coded values, which are sent to observers of the data sources. Since the amount of data sources and consequently the amount of sent messages can be specified, the load of communication on the system is controllable. Hence, different loads on the system can be tested.

4.3.2 Middleware

The middleware is the core of the blockchain-IoT application. Therefore, it is responsible to collect the sent data from registered sensor drivers. After the reception of the data, the data is prepared for further distribution. The distribution follows a two channel approach. The *real-time channel* ensures the timely distribution of the data, whereas the *integrity channel* guarantees the data's integrity.

The real-time channel uses a publish/subscribe mechanism. Thus, interested clients are able to subscribe to different data sources. The middleware sends the data to a broker which is responsible for the publication of the data and the subscriptions of interested parties. An advantage of excluding the broker from the middleware is that in case of an overload of the system, the broker can be outsourced to another server. Thus, system resources can be saved. However, the outsourcing rises new risks like trust issues with the external servers. These occurring problems have to be solved individually, when the outsourcing option is used.

The second channel is the integrity channel. The aim of this channel is to store the data in a way that the integrity can be assured. Thus, the blockchain with its tamper-proof property comes into play. The simplest way of achieving tamper-proof data, is to save the received data directly into the blockchain. Hence, the data's integrity can be guaranteed. The issue with this strategy is that storing all the data into the blockchain is not good for scaling [6, 76]. Therefore, the middleware has to reduce the data which is stored on the blockchain to a minimum. Because of that, only the data's hash is stored in the blockchain and the data itself is saved into a storage which is optimized to find the data with its hash value. This is a basic explanation of the integrity channel. The exact procedure of storing the data, adding it to the blockchain and finally distributing it to the client is presented in Figure 4.4. At first, the middleware stores the data into the storage via the storage client. The storage client is responsible for the storage process. When the data is successfully persisted, the data's hash is returned to the middleware. Similar to the broker, the storage client, as well as the blockchain client, is not part of the middleware and can be outsourced if needed. Also here, the arising risks which may occur with the outsourcing, have to be considered. When the middleware receives the hash, the middleware sends the hash to the blockchain client. The blockchain client calls a function in the smart contract by sending a transaction containing the hash. The hash is passed to the function as a parameter. The function is executed and an event is emitted. Part of the event message is the hash. The IoT client listens to the smart contract. In the background, it runs also a blockchain client, but this is not constituted in Figure 4.4. In case the smart contract emits a new event, the IoT client is notified and is able to read

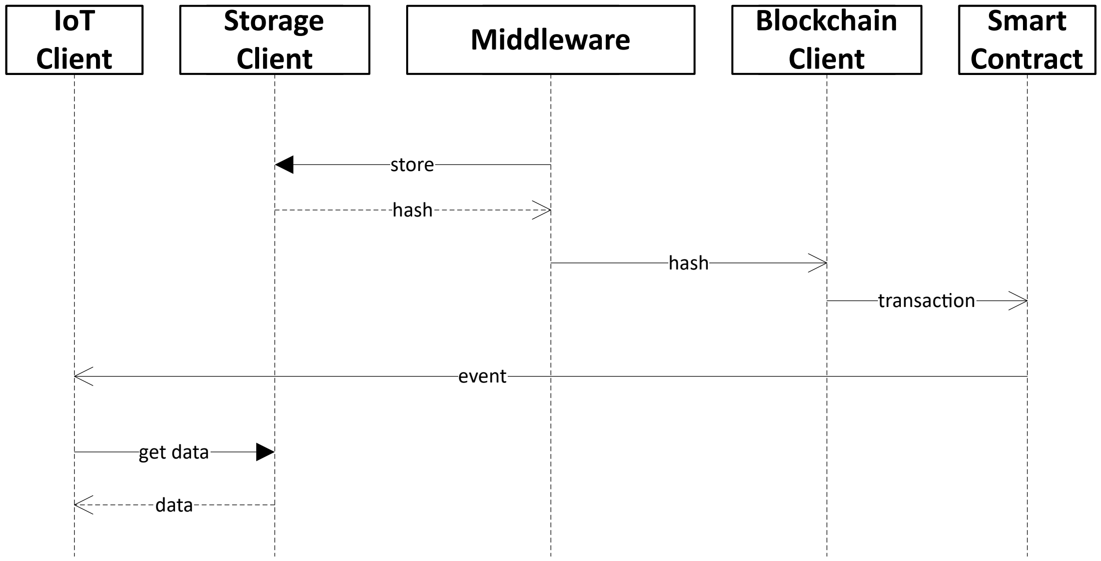


Figure 4.4: Storage of Data via Integrity Channel.

the content of the event which contains the hash. Although the events are emitted, they are also stored in the blockchain, therefore they are tamper-proof. With the hash, the IoT client can retrieve the data from a storage client. Depending on the implementation, the IoT client can share the same storage client with the middleware, but normally this will not be sensible. The reason for not sharing is, that the middleware and its storage client are supposed to run on an IoT device with restricted resources. Thus, causing more traffic to access the storage client may not be good for the performance of the IoT device. Sharing the storage client is only imaginable, when the client is outsourced to a more powerful server. After the IoT client receives the data, it can compute the hash and compare it with the received hash. In case the hashes are equal, the integrity of the data is confirmed.

4.3.3 IoT Client

As it has already been mentioned, the IoT client is responsible for tracking the performance of the two communication channels. Therefore, it is excluded from the functional and nonfunctional requirements specification. The interesting metric for the communication channel is the delay of the messages from sensing the value until the reception at the IoT client. The monitoring of the delays for the real-time channel is simple. The IoT client makes a timestamp when receiving the message and then calculates the difference between the sensing timestamp and the reception timestamp. A more detailed description of the performance evaluation process is available in Chapter 6. A little bit more complex is the monitoring of the integrity channel because several things have to be tracked. The first timestamp is taken when the IoT client is notified by the event of the smart contract. Thus, the reception of the hash is tracked. The next timestamp is taken when the data from the storage is received. Finally, a timestamp is taken when the block in the blockchain, which contains the transaction for the smart contract, is confirmed by a specified amount of proceeding blocks. This is necessary because it needs a certain amount of time or rather a certain amount of blocks until a transaction becomes immutable. The number of blocks which are needed to consider a transaction as successfully executed is six in Bitcoin and twelve in Ethereum [82]. Besides the monitoring of the communication channels, documentation of the performance results is another task of the IoT client. Therefore, the client runs only a certain amount of time until it stops listening to the channels. After that, all the results are written into a file in a format which supports further analysis in suitable programs.

Regarding the solution approach's architecture, the IoT client can be easily exchanged with another implementation of a client. This client could implement analysis features or other services which are needed for real world scenarios.

4.4 Technology Decisions

The technology decisions are made, according to the architecture of the application. The technology decisions shall support achieving the functional and nonfunctional requirements. Furthermore, efforts are made to find technology which is popular in the blockchain and IoT domain.

Regarding programming languages, Python is chosen as programming language for both drivers. The reasons for Python are the portability and the simple and fast programming. Regarding the middleware and the IoT client, a lot of functionality has to be supported. Furthermore, the middleware shall be portable and extensible. To suit these requirements, a higher programming language is searched. The decision is made for the object-oriented programming language *Java 8*[1]. Table 4.1 shows in detail the used Java packages and their usage in both implementations.

After the programming languages are set, further technology decisions have to be made. One is the communication technology, another the blockchain and the last technology decision is the storage. The details are explained in the following sections.

4.4.1 Communication

Communication plays an important role in the blockchain-IoT application because of the different interfaces between the software components. Communication is needed for the interface between the sensor driver and the middleware. Another communication technology is needed for the real-time channel. The integrity channel does not need an extra communication technology because the communication is determined by the used blockchain and storage technology. For the selection of the right communication technology, several criteria are important. First, the technology shall be well-accepted in

[1] https://docs.oracle.com/javase/8/docs [Accessed: 2018-06-08]

Table 4.1: Usage and Description of Projects Used for the Java Implementations.

Middleware	IoT Client	Name	Description
✓	✓	Maven	Build-Management-Tool
✓	✓	Jackson	JSON Library
✓	✓	Slf4j	Logging
✓	✓	Log4j	Logging
✓	✓	JUnit	Testing
×	✓	Mockito	Testing
✓	×	Californium	CoAP Framework
✓	✓	Paho	MQTT Library
✓	✓	Web3j	Ethereum Library
✓	✓	Ipfs-Api	IPFS Libary
×	✓	Commons-Cli	Command Line Library

the IoT domain. The foundation for this criterion is that the blockchain-IoT application should be extensible. Furthermore, some issues like lossy links, low power consumption or data source discovery are already handled by the technology. The last criterion is that the communication technology supports the specified design from Section 4.3. Since the communication between sensor driver and middleware as well as middleware and IoT client is rather different, it is expected that two different communication technologies are needed.

Based on the research results of Section 3.3, the communication protocols are chosen. For the interface between sensor driver and middleware, CoAP is chosen because it supports the workflow which is needed for this interface. On the one hand, it supports the request/response mechanism which is needed for the registration and, on the other hand, publish/subscribe is used for the value updates. Furthermore, it has a low power consumption and provides data source discovery. An alternative for the CoAP protocol can be a combination of two protocols. For the request/response communication, the *WebSocket* protocol can be used. The protocol enables a full-duplex, bidirectional communication between a client and a remote host [83]. For the publish/subscribe part MQTT can be used. However, one big advantage of CoAP is the data source management and the different mechanisms to interact with the data sources like observation or discovery. When using WebSockets and MQTT, an own data source management has to be implemented. REST can be the alternative to WebSockets. However, CoAP is based on REST and optimized to meet the IoT requirements. Regarding the middleware's real-time channel, MQTT is the suitable technology because it provides the needed publish/subscribe mechanism. Additionally, MQTT is optimized to operate on resource-restricted devices. Alternative technologies which also support the publish/subscribe mechanism are DDS and the *Advanced Message Queuing Protocol* (AMQP). However, DDS uses a lot of memory which is difficult for resource-restricted devices and AMQP is not suitable for real-time applications and lacks of open source libraries for resource-restricted devices [70].

To sum up, CoAP is chosen because different data sources can be added to the CoAP server and clients are able to interact with the server and consequently with the data sources in various ways. Therefore, registration, data source discovery and observation can be done in one protocol which additionally is suitable for resource-restricted devices because of its lightweight design. Regarding MQTT, it is chosen because it supports the publish/subscribe mechanism and it is optimal for resource-restricted devices. The alternatives are not suitable because of too resource-intensive requirements, lack of real-time application support or a lack of open source libraries for resource-restricted devices.

4.4.2 Integrity Channel

There are two technologies needed for the integrity channel. At first a blockchain and second a storage technology. One important criteria is that the blockchain supports smart contracts because the functionality of a smart contract is expected in the architecture of the application. The smart contract is expected to support Turing-complete languages. The second criteria is that the blockchain has to be permission-less. This is important for the integrity of the data. In case of a permissioned blockchain not everyone has access to the network [15]. Thus, not everyone is able to check the integrity of data. Therefore, the blockchain for this solution has to be permission-less. Another criteria for the blockchain selection is the maturity of the project. Since there are hundreds of blockchains and not all of them are reliable, it is important to have a blockchain with a strong developer community. If these three requirements are satisfied, the performance of the blockchain is also considered. An important criteria for the storage is if it matches the needs of a blockchain. Especially by finding the stored data with the data's hash. Since the hash is stored on the blockchain to keep the data low and to have at the same time the ability to check the integrity, a fast handling of hash keys is preferable. In addition, a P2P approach is preferred to avoid the centralization regarding the storage strategy.

Based on the defined criteria and on the research results in Section 3.2 and Section 3.4, the following technologies are chosen: Ethereum is the blockchain for the solution approach. Several reasons lead to the decision. First, it is a popular blockchain with an active community. Second, the support of smart contracts with the support of Turing-completeness is also given. Furthermore, Ethereum is a permission-less blockchain and is a lot faster than Bitcoin. According to the storage technology, IPFS is the technology of choice. Since every content is uniquely identified with its multihash, the multihash can be used to be stored in the blockchain. Additionally, IPFS follows a P2P approach which also suits into the IoT domain.

CHAPTER 5

Implementation

This chapter explains in detail how the solution approach is implemented with the chosen technologies. Therefore, class diagrams, code snippets and precise explanations of the code's functionality are used to get an insight into the implementation. The deployment of the code on the different IoT devices and the actual setup for the evaluation are part of Chapter 6. The detailed explanation of the sensor driver implementation including the virtual driver is part of Section 5.1. Section 5.2 focuses on the implementation of the middleware with its two communication channels. Finally, the IoT client is presented in Section 5.3.

5.1 Sensor Driver

As it has already been said in Section 4.3.1, there are two types of drivers. The first type reads measured values from a physical sensor and is named sensor driver. The second type is a driver with virtual data sources where the measured values are hard-coded constants. Thus, it is called virtual driver. To start the sensor driver, the command, shown in Listing 5.1, has to be executed. Several options have to be set. Option *a* indicates the id of the sensor driver. This id shall be unique to identify the driver. Thus, the *Internet Protocol* (IP) address including the port is suitable. The reason why the IP address is not looked up in the code, is that a device can have several IPs, for instance, *Internet Protocol Version 4* (IPv4) and *Internet Protocol Version 6* (IPv6). Because of that the id is set manually to decide which IP shall be used. The option *r* specifies the port of the CoAP server which is started in the code. This server is used to handle the observation requests and consequently the notification of the middleware. The IP address of the CoAP server is hard coded and set to "0.0.0.0". The next options *s* and *p* indicate the IP address of the middleware's registry server and the port.

Listing 5.1: Sensor Driver Start Command

```
USAGE: hum_temp_sensor_driver.py −a <ip address with port> −r
   <port for own coap server> −s <server ip address> −p <port>
```

Listing 5.2: Virtual Sensor Driver Start Command

```
USAGE: virtual_driver.py −a <ip address with port> −r <port
   for own coap server> −s <server ip address> −p <port> −c
   <number of data sources>
```

Listing 5.2 shows the command to start the virtual driver. As it can be seen, there is no difference to the sensor driver command except for the additional option *c*. The option *c* specifies how many data sources shall be created in the virtual driver. The specified data sources in the virtual driver are basically implemented like the sensor driver's data sources. The only difference is that the data sources of the virtual driver are not interacting with a sensor.

The implementation of the sensor driver collects values from the Grove Temp&Humi Sensor[1]. The sensor is connected to the Raspberry Pi with the GrovePi+[2] bridge. An advantage of this setting is that the sensor can be connected easily with the Raspberry Pi and the Python library *grovepi* is provided by the producer which makes it handy to retrieve values from the sensor. Since only one command is needed to retrieve values from the sensor with the *grovepi* library, the main effort of the driver implementation is the realization of the registration at the middleware and the notification of subscribed clients. To achieve this, the driver has to work as a client for the registration and as a server to receive observe requests and to push notifications. The used library for the CoAP protocol is *coapthon*[3]. The first task which is done by the sensor driver after the start, is the parsing of the command line arguments. After that, a CoAP client is started. The client sends a *POST*-request to the registry server of the middleware. As payload of the request, the address of the sensor driver's CoAP server is transmitted. The address scheme is "coap://<ip:port>". When the registration is successful, the client is closed and the CoAP server is started. Different data sources, where every data source has its own path, can be added to the server. Every data source in the sensor driver represents a sensed phenomenon. In case of the virtual driver, no phenomenon is sensed. The added data sources only simulate a phenomenon whereas the amount of added data sources depends on the command line argument *c*. The specified amount is added to the server in a loop. In case of the Grove Temp&Humi sensor, two data sources are needed because the sensor is able to sense temperature and humidity. The relative paths for

[1] http://wiki.seeedstudio.com/Grove-TemperatureAndHumidity_Sensor [Accessed: 2018-06-13]

[2] https://www.seeedstudio.com/GrovePi%2B-p-2241.html [Accessed: 2018-06-13]

[3] https://github.com/Tanganelli/CoAPthon [Accessed: 2018-06-19]

every data source are hard-coded. In case of the Grove Temp&Humi sensor the sub-paths "/temperature" and "/humidity" are chosen. The names describe the data sources or rather describe what the data sources are measuring. Since the full path including the IP of the driver is used for the identification of the sensors, a meaningful naming of the data sources helps for the semantic understanding.

A data source is implemented as a Python class, which extends the *Resource* class of the *coapthon* library. To enable the observe option, the boolean value *observable* has to be set to true in the constructor. The observe option can be included in a GET request [84]. Thus, the GET method of the *Resource* class has to be overwritten. When GET is called on the data source's path, a message in *JavaScript Object Notation* (JSON) format, which contains the current measured value, is returned. If a client calls GET with the observe option, not only the message is returned, but the client is also added to the list of observers. To continuously notify the clients, which are contained in the list of observers, a method is implemented which calls itself within a defined interval of five seconds. In every call, the current value of the sensor is collected. Regarding the virtual driver, only a hard-coded value is assigned. When the value is set, the CoAP server is notified. The server takes over the control of distributing the new values to the clients, which are added in the list of observers. The virtual driver has a limitation of 60 sent messages per data source. This fixed amount of sent messages shall help in the analysis phase to detect message losses. To find out, which data sources are available at a certain driver or rather CoAP server, clients can send discover requests. In response to the request, a list of links of the available data sources is returned. A more detailed explanation of the discover functionality is part of Section 5.2.

Two things have to be mentioned concerning the observe option. The first thing is the five second notification interval. In a resource-restricted environment, every message which is not sent, saves resources, for instance bandwidth. Thus, another strategy to send the values is that the server and consequently the clients are only notified when the value changes. Another possibility can be the notification of the clients when a certain threshold is exceeded. The disadvantage of a strategy which does not deliver values continuously, is that a connection loss is less obvious. To distinguish a connection loss from a long period of no value change, timeouts can be set. However, this needs a certain amount of time until the timeout comes into play. Furthermore, this timeout has to be propagated through all application levels properly. Hence, the risk is omnipresent that the connection loss is never propagated to the user of the application. It can be assumed that the values are part of the application's feature, thus they are used somewhere. When a continuous delivery of values is expected, the absence of new values is obvious. A very simple example is when temperature is measured and displayed on a screen. In the continuous delivery case, every n seconds the value is refreshed on the display. The refreshing is visible to the user, hence, the absence of a new value can also be visible by setting an error value, if no value is delivered. In the noncontinuous case, the displayed value would not change and if the timeout is not propagated always the same temperature is displayed. Therefore, it can not be said with certainty if the temperature is constant

Listing 5.3: JSON Message from Driver to Middleware

```
{
value: <value>,
unit: <value>,
timestamp: <value>
}
```

or if there is a connection problem. Besides the connection argument, the strategy is also dependent on the sensed phenomena. Continuous notifications for a fire sensor make less sense than for temperature. In general, it can be said that the choice of a notification strategy is dependent on the sensor and on the preferred strategy. Therefore, the implementation of drivers provides a lot of flexibility to meet the requirements. For the performance evaluation, the continuous delivery is chosen because it is more predictable and has a constant load of sent messages. The second consideration in the driver implementation is the message format for the exchange between the driver and the middleware. Listing 5.3 shows the message format which is used for the exchange. Also for the format, the resource-restricted argumentation has to be considered. The first idea to minimize the message size is to only send the value. However, only sending the value is not desired because the interpretation of the value is hard without further information. Thus, context-awareness is key in IoT [9]. Concatenating all the meta-information into one string is also tricky in terms of interpreting the string. For the interpretation there has to be a standard which defines the positions for certain information. Using an already existing structured format is more sensible. Therefore, JSON is used for the exchanged messages. Another advantage of JSON is that it is supported by various libraries for various programming languages. Regarding development reasons, the messages can be easily extended with further information if needed. As it is constituted in Listing 5.3, the meta-information is kept at a necessary minimum. Since the type of the measurement can be derived from the data source path, it is not added to the message format. The unit of the measurement has to be added because one phenomenon can have different units. For instance, temperature can have degree Celsius or Fahrenheit. The timestamp is to uniquely identify the measured values in combination with the data source path. Additionally, this timestamp is used to calculate the delays for the performance evaluation.

To sum up, it can be said that the implementation of the driver offers a lot of opportunities for the blockchain-IoT application. Already on this low level a lot of design decisions can be made. Furthermore, optimizations like notification frequency or additional functionality, like dynamic adding of sensors to the middleware, can be implemented. Regarding the virtual driver, it can be seen that the differences to the sensor driver are minimal, hence the usage of the virtual driver is a legit option to test different sensor loads. Figure 5.1 summarizes the explained workflow, on the example of the virtual driver, in a sequence diagram. However, the sequence diagrams only shows the case with a successful registration at the middleware. Additionally, the data sources are called

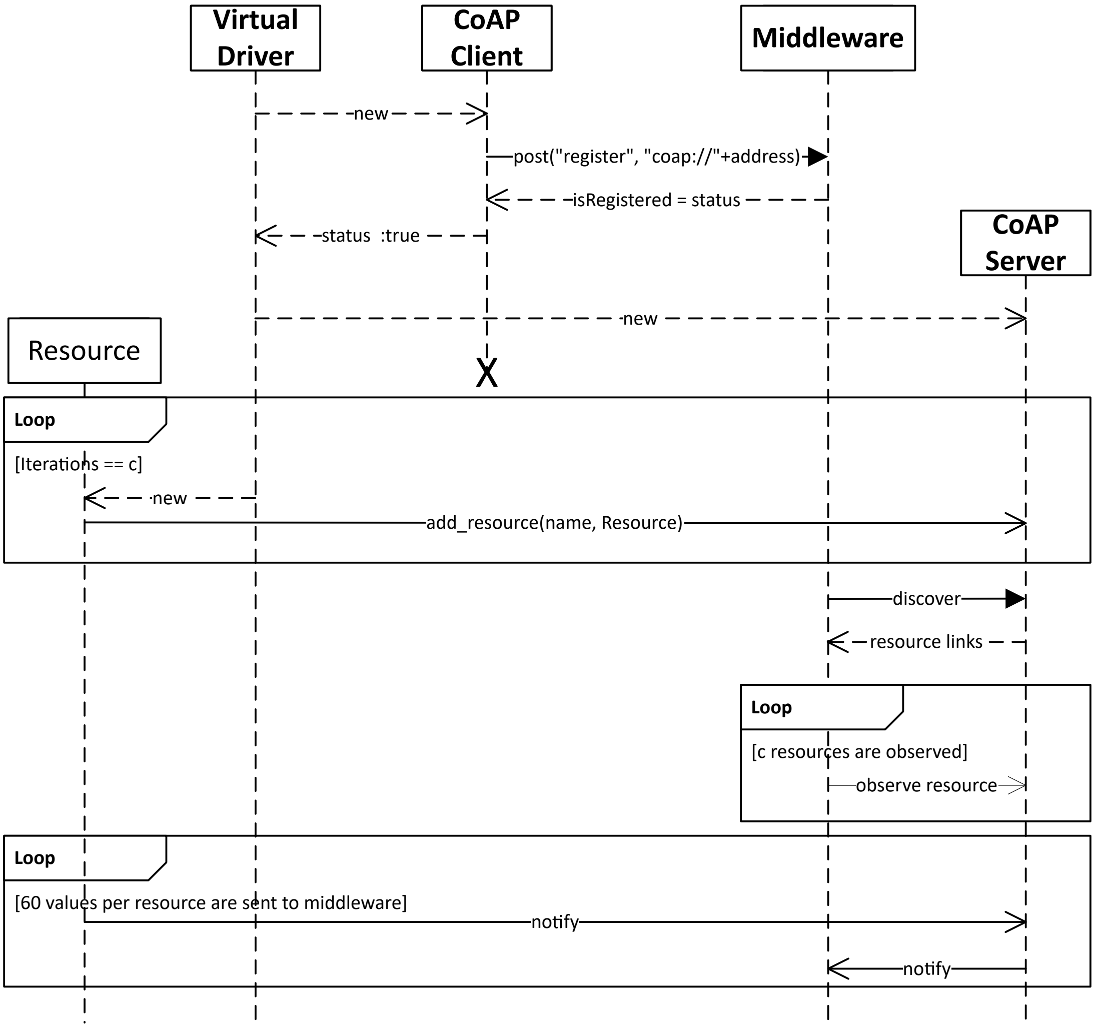


Figure 5.1: Workflow of the Virtual Driver in Case of a Successful Registration.

resources in this figure because in the sensor driver the data sources are implemented with the *Resource* class.

5.2 Middleware

The middleware as the core of the application is responsible to transmit data from the drivers to the IoT clients. In general, it can be said that the middleware is an event-driven application. Event-driven applications continuously interact with their environment [85]. Furthermore, incoming events are processed and the corresponding tasks are executed. For instance, a button press or a notification can be an event. The exchange of the values between driver and middleware is push-based, since the sensor driver pushes the

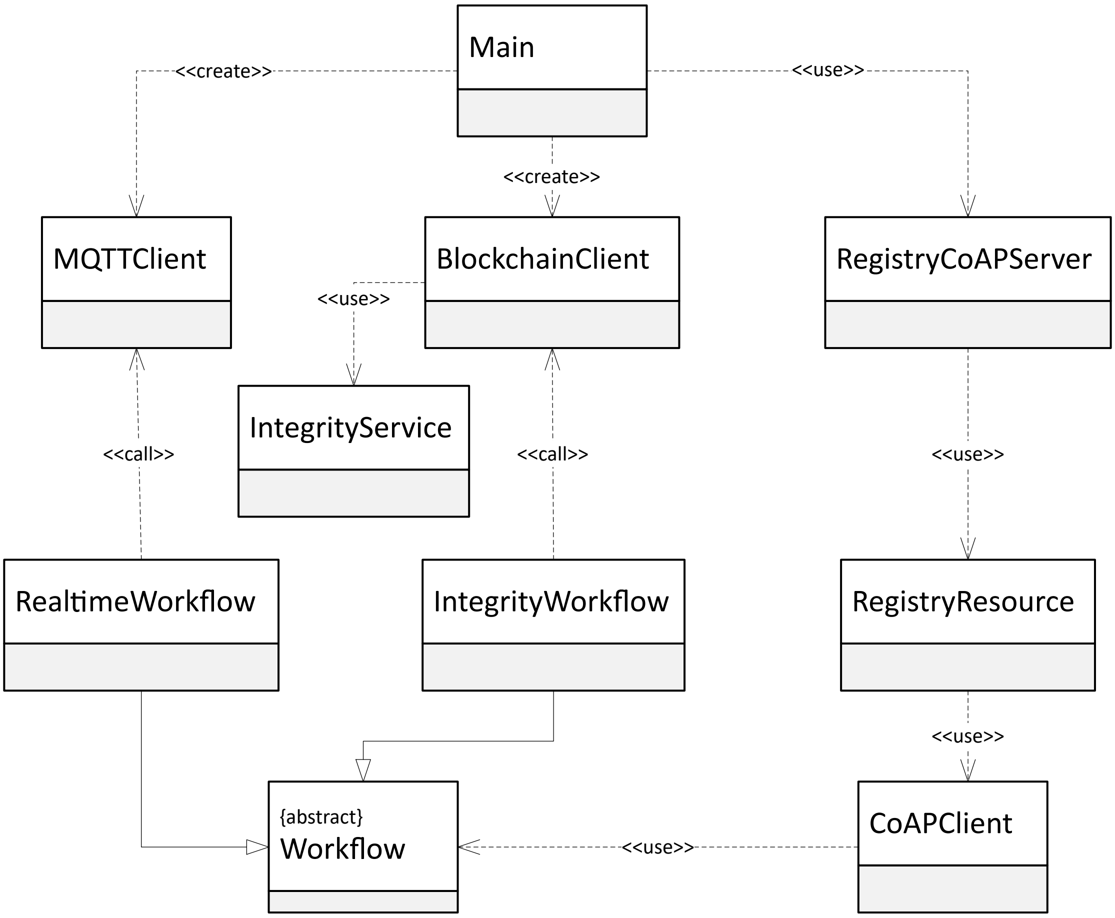


Figure 5.2: Middleware Class Diagram

notification to the clients from the list of observers. The architecture of this event-based application is constituted in Figure 5.2. Before the discussion of the classes starts, two classes have to be presented which are not part of Figure 5.2, but they are used in the middleware and in the IoT client.

The *Message* and *Measurement* classes are a representation of the JSON messages which are sent between driver and middleware as well as middleware and IoT client. Figure 5.3 shows how the classes are aggregated. The conversion from JSON to a Java object and vice versa is done with the extra helper class *JsonConverter*. For the conversion, the Jackson[4] JSON library is used. *Measurement* is used for the messages of the sensor and virtual driver. For the internal use in the middleware and for the further distribution, *Message* is used. *Message* contains not only the measurement results but additional information about the device where the values come from. This is necessary to determine the origin of the data when it is propagated to different stakeholders. That means as

[4] https://github.com/FasterXML/jackson [Accessed: 2018-06-12]

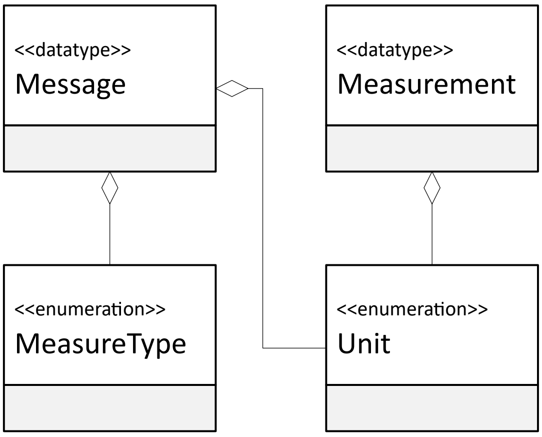


Figure 5.3: Message Objects

soon as *Measurement* is received in the middleware, a *Message* object is created and used for the distribution over the different communication channels.

The *Main* class is the entry point for the middleware. It takes one optional argument *m*. This determines the maximum inflight value for the *MQTTClient*. If the option is not set, a default value is used. Further details are discussed in Section 5.2.1. After the reading of the argument, the connections to the MQTT, blockchain and IPFS client are tested by starting a connection. For MQTT, the *MQTTClient* and for the blockchain the *BlockchainClient* is used. Regarding IPFS, the according class of the IPFS library is used. In case of problems with the connections, these problems are detected at an early stage and the middleware terminates. If the three connections are working, the *RegistryCoAPServer* is started. All further activities of the application are started via the *RegistryCoAPServer*, when new registry requests of drivers are received.

The *RegistryCoAPServer* contains the *RegistryResource*. The *RegistryResource* is accessible via the path "/register" and implements a POST method. Incoming POST-requests are expected to contain the address of their CoAP server as payload. The driver is registered by creating a new *CoAPClient* which is initialized with the received address. After the creation, a new thread is started to run the *CoAPClient*.

The *CoAPClient* is the class which connects the driver with the middleware. For the actual CoAP communication with the driver, the *CoapClient* of the Java CoAP framework Californium is used by the *CoAPClient*. However, in favor of describing the workflow in an understandable way, the communication is described from the perspective of the implemented *CoAPClient*. Figure 5.4 constitutes the workflow of the *CoAPClient*. Although the constituted driver in the figure is the virtual driver, it would make no difference to name the sensor driver. After the creation of the *CoAPClient* by the *RegistryResource*, the run-method is invoked. As soon as the run-method is invoked, a connection to the sensor's CoAP server is established. In the next step, the CoAP *discovery* command is executed. There are two types of discovery in CoAP [86]. One is the

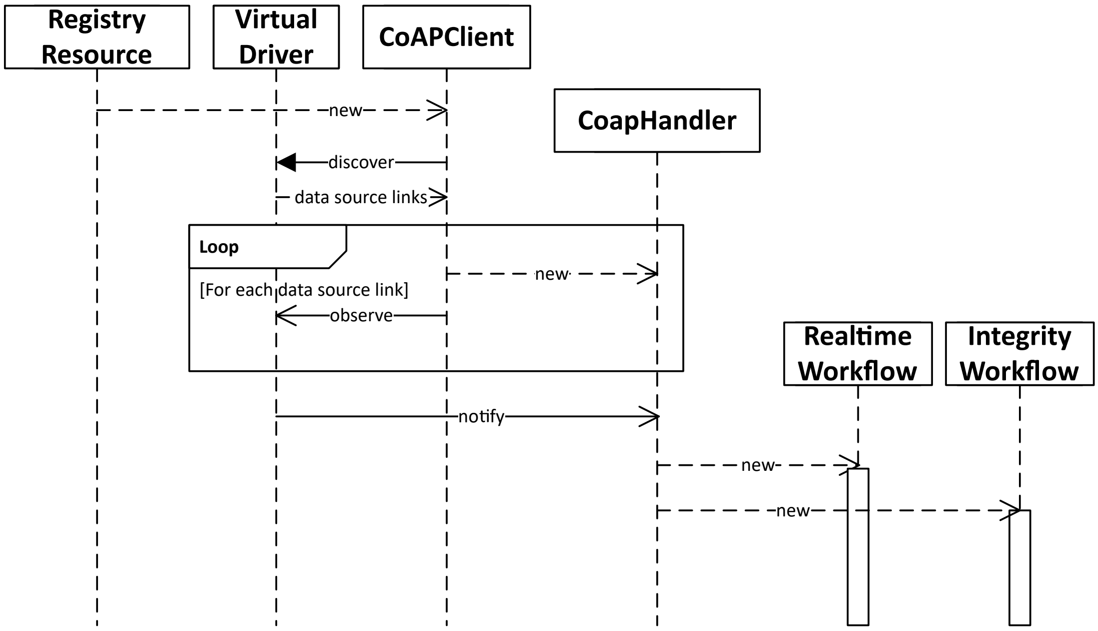


Figure 5.4: Initialization and Observation of a New Virtual Driver.

service discovery, the other is the resource discovery. The term resource equals the term data source of this thesis. The resource discovery is important in M2M applications where no humans are involved because the discovery function increases the interoperability. To get all available data sources of the driver, the middleware sends a discovery-request. A list of the data sources' links is returned. The list is iterated and for every link a *CoapHandler* object is created. Furthermore, an observe-request is sent to the driver and the created *CoapHandler* is used as a callback object for incoming notifications. The *CoapHandler* object contains two methods, the *onLoad*- and the *onError*-method. In case of an error, the *onError*-method is called. In case of a notification from the driver's CoAP server, the *onLoad*-method is called. Figure 5.4 only shows the case where a notification is sent by the driver and the *onLoad*-method is called. The *onLoad*-method parses the payload which is in JSON format and creates a *Measurement* object. Since the information of the JSON message or rather the *Measurement* is too minimal for further usage, a *Message* object is created. After that, the further processing of the data is defined by passing the *Message* object to different *Workflow* objects. Every distinct *Workflow* object defines an independent processing of the data. In the current implementation *RealtimeWorkflow* and *IntegrityWorkflow* are used, which are subclasses of *Workflow*. The *Workflow* class defines a constructor which takes a *Message* object as input. For every new *Message*, a new *Workflow* has to be created. Regarding the further development of the middleware, other workflows can be defined to implement different processing strategies for the data. Theoretically, it is possible that workflows contain other workflows. For example, the super workflow can filter events and pass the filtered events to the subworkflow. As long as the CoAP server is delivering new values, the *CoAPClient* is executed. To enable constant listening to new incoming messages, the two workflows *RealtimeWorkflow* and *IntegrityWorkflow* are executed in separated threads. For a resource-efficient handling of the threads, an *ExecutorService* is used. This *ExecutorService* is also used for the threads of the *CoAPClient*. The *ExecutorService* is initialized with a cached thread pool which means that the size of the thread pool increases when new threads are needed. The risk of this pool is that too much memory is used, hence the application would crash. To solve this, the amount of concurrent threads shall be limited. Nevertheless, in this implementation the amount of threads is not limited because it can not be said what the maximum amount is. Since the aim is to evaluate the performance, the cached thread pool does not limit the amount of threads and therefore the devices can be tested to their limits. In case this middleware would be used in a productive application, the threads shall be limited to an amount, where the system is stable but the performance is satisfying.

5.2.1 Real-Time Channel

The real-time channel comprises the classes *RealtimeWorkflow* and *MQTTClient*. Besides the implementation of the classes, a broker is needed which implements the MQTT protocol. Thus, *mosquitto*[5] is installed which is a lightweight broker and suitable for resource-restricted devices. The communication with the broker is done via the *MQTTClient* class.

The *MQTTClient* connects to the broker and publishes payload to the broker under a certain topic. The connection is initialized with a maximum inflight value by setting option *m*. The maximum inflight value determines the number of messages, which can be transmitted by the broker concurrently with a QoS-level of one or two [87]. In case option *m*, which is parsed in the *Main* class, is set, option *m*'s value is used else a default value is used. The content, which is published to the broker, has a QoS-level of two. A QoS-level of two means that the messages are sent exactly once [71]. The reason for this high QoS-level is that the delivery of the sensor data shall be reliable. Thus, the subscriber of certain topics does not have to implement mechanisms which detect data losses or duplicates, if the reception has also a QoS-level of two.

The *RealtimeWorkflow* defines the workflow for the real-time channel for one *Message* object. When the *RealtimeWorkflow* is executed, the *Message* is converted to a byte array. Then, the byte array and the topic are passed to the *MQTTClient*. The topic for the payload is the id of the data source from where the message comes. Since the id of a data source equals a topic, it is easy for a subscriber to exactly choose the data sources from which the values shall come. As it can be seen, the logic of the real-time channel is not complex. When the right communication protocol is chosen, there are little implementation efforts needed. Nevertheless, messages are delivered in a reliable way and the broker provides a well-defined interface to the data sources' messages.

[5] https://mosquitto.org [Accessed: 2018-06-24]

5.2.2 Integrity Channel

The integrity channel comprises the classes *IntegrityWorkflow*, *BlockchainClient* and the *IntegrityService*. This channel distributes the data via the Ethereum blockchain in combination with IPFS. To access the blockchain, a blockchain client must be started. The used client for Ethereum is *go-ethereum* with the command line interface *geth*[6]. To save resources, the client is executed with the light client protocol. The light client protocol [78] is developed for the usage of Ethereum in low-capacity environments. The protocol downloads the block headers and verifies only a little bit. Thus, approximately one kilobyte of data every two minutes is processed. Using the light client protocol has not the full security of a full node which disposes of the full information. However, the light client is able to verify the execution of transactions and has a high-security assurance about the current state of the Ethereum blockchain at least of some particular parts. Since the light client only downloads the block headers, special mechanisms are implemented to meet the use cases of a light client, for instance, watching for logged events. There are also other use cases like checking for transaction confirmations, but the event use case is relevant for the integrity channel. Hence, the watching of logged events is explained. The light client receives the block headers which are searched for *Bloom filters* matching addresses or certain topics of interest. A Bloom filter is a probalistic data structure [88]. In general, a probabilistic data structure uses probabilistic approaches, approximation principles and hashing to achieve a fast processing of data. Therefore, these data structures are suitable for big data and streaming applications. The Bloom filter is a representation of a set such that space usage is low. Only two operations are supported, namely insertion of elements and checking if an element is in the set. The accuracy of the filter depends on the size of the set and on the number of used hash functions. When the block header is matching, all the transaction receipts are downloaded. A transaction receipt is an encoding of a transaction where certain information is contained, regarding the transaction's execution [19]. Among other things, Bloom filters composed from the logs' information are part of a transaction receipt. The receipts are checked for Bloom filter matches. After finding a match, the light client checks the transaction receipt's log for a match. This is how a light client is looking for an event. Generally, the whole light client protocol is still under development. The actual implementation of the protocol is the *Light Ethereum Subprotocol* (LES) [89]. Version 2 is the current version. To avoid conflicts with the main Ethereum protocol and for a more independent development, the light client protocol is designed as an extra protocol. It supports the download of the block headers and several on-demand data retrievals for further information. Full nodes can support LES, hence this functionality is served to light clients.

To start the light client, the command shown in Listing 5.4 is used. The *testnet* option indicates that the client shall connect to the Ropsten testnet [90]. *rpc* starts a *Hypertext Transfer Protocol* (HTTP)-*Remote Procedure Call* (RPC) server where an application can connect. To enable the light client protocol, *syncmode* has to be set to *light. datadir*

[6] https://ethereum.github.io/go-ethereum [Accessed 2018-06-22]

Listing 5.4: Start of Geth in Light Client Mode

```
geth --testnet --rpc --syncmode=light --datadir=./ropsten
   --unlock 0x7f72bfcb5946b51b8e43ffa43e0310f1c6b84bf8
   --password pwd.txt
```

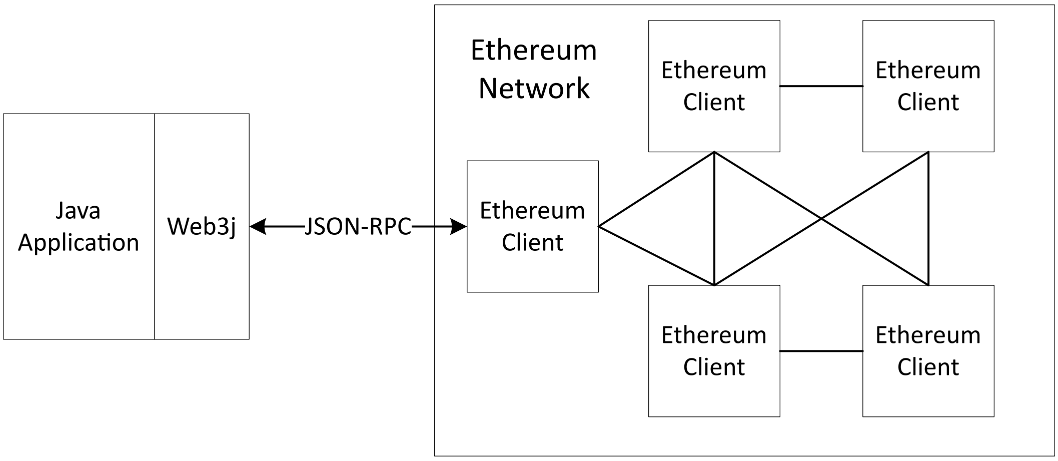


Figure 5.5: Web3j and the Ethereum Network. (Source: [91])

indicates the directory where all the blockchain data has to be stored. To send a transaction, the issuing account has to be unlocked. Thus, the *unlock* option specifies the account which shall be unlocked and *password* takes a file as argument. The password for the account is read from the file. When the client is started, the synchronization of the block headers starts as soon as other nodes are found.

To connect a Java application to a running client, Web3j can be used. Web3j is a Java and Android library which implements the JSON-RPC API of Ethereum [91]. The JSON-RPC API is used to interact with the Ethereum client [92]. Therefore, JSON is used as a lightweight data-interchange format. JSON-RPC is the protocol to call different methods on the Ethereum client like *eth_syncing* which returns the current synchronization status. Figure 5.5 constitutes how the Web3j library links the Java application with the Ethereum network. It can be seen that Web3j needs a running instance of an Ethereum client. Otherwise, Web3j can not connect to the network. Two different Ethereum clients are supported by Web3j. One client is the Geth client and the other is the Parity[7] client. With Web3j the connection to the client can be established via HTTP or *inter-process communication* (IPC) [91]. Furthermore, Web3j enables to interact with smart contracts via contract wrappers, which are Java classes representing the contracts. Therefore, the interaction with the smart contract is like operating with a Java class.

[7] https://www.parity.io [Accessed: 2018-06-25]

The *BlockchainClient* is the class in the middleware which does the communication with the Ethereum client. The class implements two methods to interact with the client. The first method is the establishment of the connection. In this method, a connection is established to the HTTP-RPC server of the client. Furthermore, the smart contract is created by creating a new object of the smart contract wrapper class *IntegrityService*. The second method calls the *update*-method in the *IntegrityService*. That means that this method issues a transaction in the blockchain via the client. The *update*-method is the only method of the smart contract which is used by the middleware whereas the smart contract implements further methods.

The smart contract and its according wrapper class *IntegrityService* implement several methods to achieve the defined functional requirements such as access of data with guaranteed integrity, automatic check of update frequency and penalty payments. The smart contract is implemented with Solidity[8]. Furthermore, the aim of the implementation is to roughly sketch the motivational scenario, which considers the monitoring of environmental data. Thus, two parties are assumed, which are involved in an agreement in form of the smart contract. One is the institution A and the second is the client B whereas in the motivational scenario the institution would be the European Commission and the client one of the Member States. The institution A creates the smart contract and sets the maximum delay. This maximum delay determines the time period in which the updates have to be delivered. The second parameter, which is set by A, is the deposit. Client B has to pay the deposit during the registry on the smart contract. This deposit ensures that money is available on the smart contract, which is necessary when client B misses to deliver data in the specified time period and a penalty has to be paid. When institution A has created the smart contract, client B has to register on the smart contract. The registration of the client is not done implicitly by A because the deposit has to be paid. Thus, B has to register himself explicitly and pay the deposit. The *register*-function can only be called once if a registration is successful. With the registry the deposit is transferred from the account of client B to the smart contract. Additionally, the address of B is saved in the contract. In case the amount of the transferred deposit is not correct, the changes to the state are undone and the client has to repeat the registry.

The core function of the smart contract, which checks the update rate, calculates penalties and distributes the data, is the *update*-function of Listing 5.5. The first thing to mention is the function modifier *onlyBy(client)* in line 8. This function modifier controls the access of the function. In this case, only the registered client B is able to access the function. The *update*-function contains several parameters. *function_code*, *digest_length* and *digest* are the parts which form the multihash of IPFS. The parameter *id_hash* is the hash of the data source's id. The identification of the data source is important to track the update rates of every single data source. Therefore, every timestamp of the data source's last update is stored in a structure called *mapping*. This data structure maps a certain key to a value. When *update* is called and the last update timestamp plus the maximum delay is less than the current timestamp, which is checked in line

[8] http://solidity.readthedocs.io/en/v0.4.24/index.html [Accessed: 2018-06-22]

Listing 5.5: Update-Function in Smart Contract “IntegrityService”

```
1   event MeasurementUpdate(
2           address indexed sender,
3           uint8 function_code,
4           uint8 digest_length,
5           bytes32 digest
6       );
7
8  function update(uint8 function_code, uint8 digest_length,
      bytes32 digest, bytes32 id_hash) onlyBy(client) public {
9
10          require(registered == true && penaltyPaid == false);
11
12          if (lastUpdates[id_hash] > 0 && block.timestamp >
               lastUpdates[id_hash] + maxDelay) {
13              penalty += 1;
14          }
15
16          lastUpdates[id_hash] = block.timestamp;
17          emit MeasurementUpdate(msg.sender, function_code,
               digest_length, digest);
18      }
```

12, a penalty is calculated in line 13. In this implementation the penalty is only one *Wei*, which is the smallest unit of currency in Ethereum [93]. One quintillion Wei are one *Ether*. The small amount is chosen because the penalty is not in the focus of the thesis. The aim of implementing this penalty mechanism is to show how this can be realized. However, before the penalty is checked there is another check in line 10. Solidity provides *require* to check conditions [94]. In case the condition is not true, an exception is thrown and all changes on the state are undone. The *require* command is also used in the implementation of the function modifier *onlyBy(client)* and in the *register*-function. In the *update*-function it is used to check, if the caller of the function is registered as client B and if the penalty is already paid out to the institution A. When the penalty is paid out, no further updates are accepted, hence this smart contract will not be used anymore. The reason for a defined ending of the contract shall provide a limitation to the validity of the agreement between client B and institution A. Additionally, the deposit does not cover penalties for an infinite amount of time, hence a limited amount of time makes it more predictable that the deposit is high enough to cover potential penalties. After the execution of lines 10 to line 16, line 17 is executed. This line is important for the distribution of the data to the clients. By calling the *emit*-command in line 17, the event *MeasurementUpdate* is emitted. Events use the logging facilities of the EVM [95]. These logging facilities can be used to create callbacks in applications which listen to the events. The events are stored in the transaction's log whereas the logs are associated with the smart contract which emitted the events. Parameters of the event can have the attribute *indexed*. Thus, these parameters are not stored themselves, but it is possible to search for the parameters and filter them in the user interface. All parameters, which have not the attribute *indexed*, are stored in the data part of the log. The *MeasurementUpdate* event contains four parameters. The first parameter, in line 2, is an indexed parameter and stores the address of client B because B is the caller of the *update*-function. The idea behind it is, in case all measured values of one address shall be found in the blockchain, it can be filtered by the indexed parameter. The other three parameters, in the lines 3 to 5, are the three parts of the multihash. When a listening client receives the event, the client is able to build the multihash and retrieve the data from IPFS. That means the emitted events are used to distribute the data to the clients. For the performance evaluation, the *update*-function is the only part of the smart contract which is used, since the aim is to compare the delivery speed to the real-time channel. To complete the sketch of the motivational scenario, also disburse methods are implemented. Before client B can withdraw the remaining deposit, the institution A has to withdraw the penalty. Since the mapping is not iterable without a data structure on top of it [96], institution A has to call the *getLastUpdate*-function for every data source of client B. This function returns the timestamp of the given data source's id hash. Additionally, the current penalty is calculated during the call. This penalty calculation is necessary, when a data source has not only skipped a few value deliveries but stopped delivering values. When the delivery of values is stopped, the *update*-function can not update the penalties. Hence, the explicit calculation is required. The disadvantage of this method is that the institution has to keep track of all data sources. Nevertheless, the complexity of the smart contract is reduced. When the institution has updated the penalty by calling the *getLastUpdate*-function for every data source, the *withdraw*-function is called and the penalty is transferred to the account of A. After that, client B is able to withdraw the remaining deposit ($deposit - penalty$).

As it has already been explained, IPFS is needed, besides the blockchain, for the distribution of the data. Similar to the blockchain, IPFS also needs a client to access the file system. Therefore, the IPFS client *go-ipfs*[9] is installed which provides a HTTP API for the controlling of the node. Before starting the client, it has to be initialized with a profile. The predefined profile *lowpower* is used. This profile shall reduce the overheads of the client [97]. To connect the middleware with the client, the *java-ipfs-api*[10] is used. Another thing which has to be mentioned is the storage mechanism of IPFS. The data in IPFS is stored in someone's local storage as objects [74]. When an object is requested, it has to be found, downloaded and stored locally whereas the object is stored temporarily. When the object shall be stored permanently, it has to be *pinned*. That means in case of the middleware that the device has to be available all the time to provide the measured sensor data. With this setting the P2P properties of IPFS can not be used because

[9] https://dist.ipfs.io/#go-ipfs [Accessed: 2018-06-23]

[10] https://github.com/ipfs/java-ipfs-api [Accessed: 2018-06-23]

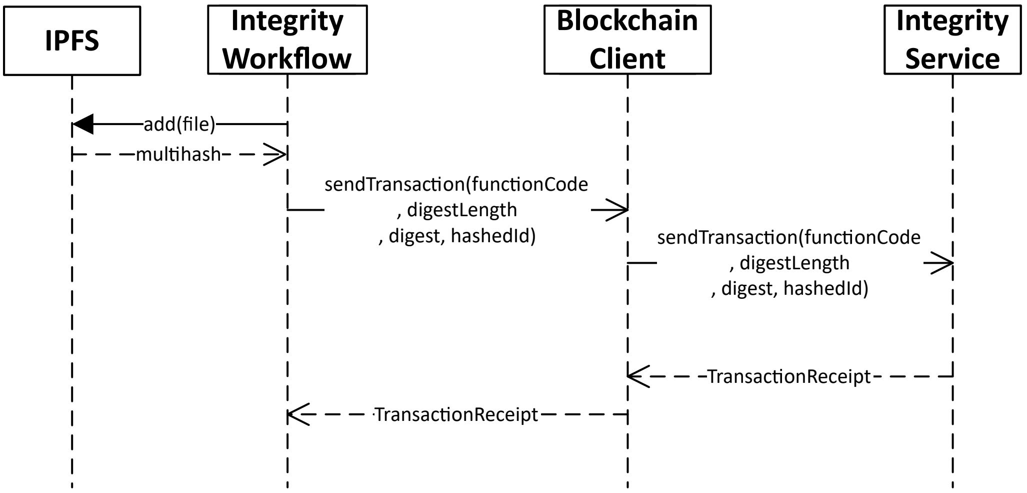

IPFS
Integrity Workflow
Blockchain Client
Integrity Service
add(file)
multihash
sendTransaction(functionCode , digestLength , digest, hashedId)
sendTransaction(functionCode , digestLength , digest, hashedId)
TransactionReceipt
TransactionReceipt

Figure 5.6: Distribution of Message via Integrity Channel.

the IoT device is a single point of failure. One solution is the IPFS Cluster[11] which supports the allocation, replication and the tracking of Pins within a cluster of IPFS clients. Another solution is the Filecoin which is a token running on a blockchain [98] where miners earn the token by providing storage. Although there are solutions for the single point of failure, it is not realized in this implementation because data availability is not in the scope of this thesis.

The *IntegrityWorkflow* contains the logic for the whole integrity channel. Thus, the blockchain and IPFS clients are needed. Figure 5.6 constitutes the steps, which are executed to send a message via the integrity channel. When the *IntegrityWorkflow* is started by calling the *run*-method, which is done via the *ExecutorService*, the connections to the blockchain and IPFS are established. After that, the *Message* object of the *IntegrityWorkflow* is converted to a JSON byte array and added to the IPFS network. This operation returns the multihash, which is divided into the distinct parts function code, digest length and digest. Furthermore, the id of the *Message* is hashed with *SHA-256*. With these four parameters, the *sendTransaction*-method, which passes the parameters to the *update*-function in the *IntegrityService*, is called in the *BlockchainClient*. A *Future* object is returned where the transaction receipt is returned as soon as it is available. This *Future* can be used for exception handling. However, in this implementation the result is only logged. The reason for that is the life cycle of a transaction. Figure 5.7 shows the different states of a transaction. First, the transaction is submitted to the transaction pool, where all miners can choose the transactions which are included into a block [82]. When the mined block is confirmed by eleven subsequent blocks, the transaction can be seen as committed. Hence, the transaction remains in the block with a high probability. If the transaction is included in a block, which is part of a shorter chain, then the transaction is returned into the transaction pool. The rate of transactions,

[11] https://github.com/ipfs/ipfs-cluster [Accessed: 2018-06-23]

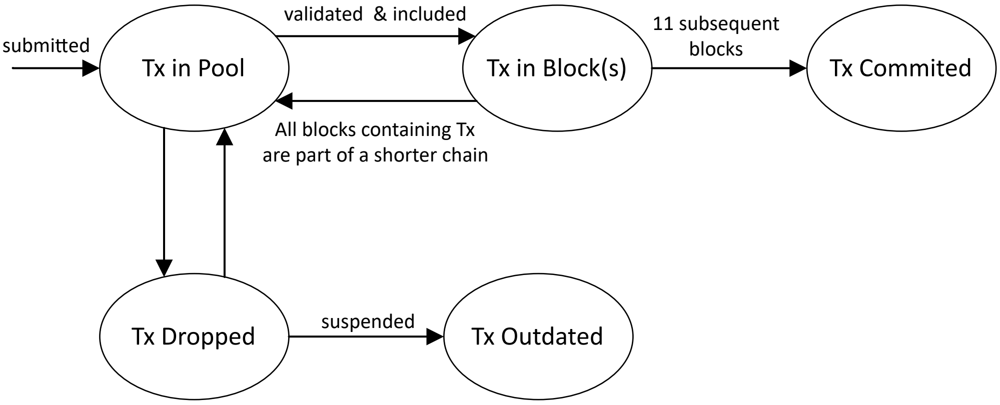


Figure 5.7: Life Cycle of a Transaction. (Source: [82])

which are returned into the transaction pool after the first inclusion, is 0.021%. After the third inclusion it is only 0.000007%. Since the expected rate of transactions, which are not included in the blockchain, is very low, there is no exception handling realized in the current code of the middleware. Additionally, the low rate can be ignored for the performance analysis because it has no relevant impact. If a transaction is dropped or outdated, it can not be determined with certainty because the invalid transactions can become valid in later states of the system. Furthermore, dropping the transaction is a local decision of a miner, hence it is not possible to know with certainty if every node in the network has dropped the transaction. The only case where a transaction is certainly outdated for all nodes, is when a new transaction from the same account with the same nonce is submitted. Thus, the transaction is definitely invalid and will not be included in a block. The uncertainty of the states makes an exception handling difficult. However, unless a small rate of uncertain transactions is tolerable, an exception handling shall be implemented.

5.3 IoT Client

The IoT client is used to measure the delays of the different communication channels and writes the results in a *Comma Separated Values* (CSV) format. This format shall enable a further analysis of the results. As it has already been said in Section 4.4, Java is used for the implementation of the IoT client. Figure 6.1 presents the architecture of the IoT client.

The *LogMessage* class with its subclasses *RealtimeLog* and *IntegrityLog* is used to store the information about the delays and further log information during the execution. Since the log information of the real-time channel is different from the log information of the integrity channel, there are two subclasses. After the termination of the subscription to the channels, the *LogWriter* writes the collected *LogMessage* objects into a CSV file. A sensor value is distributed twice, once through the real-time channel and a second time

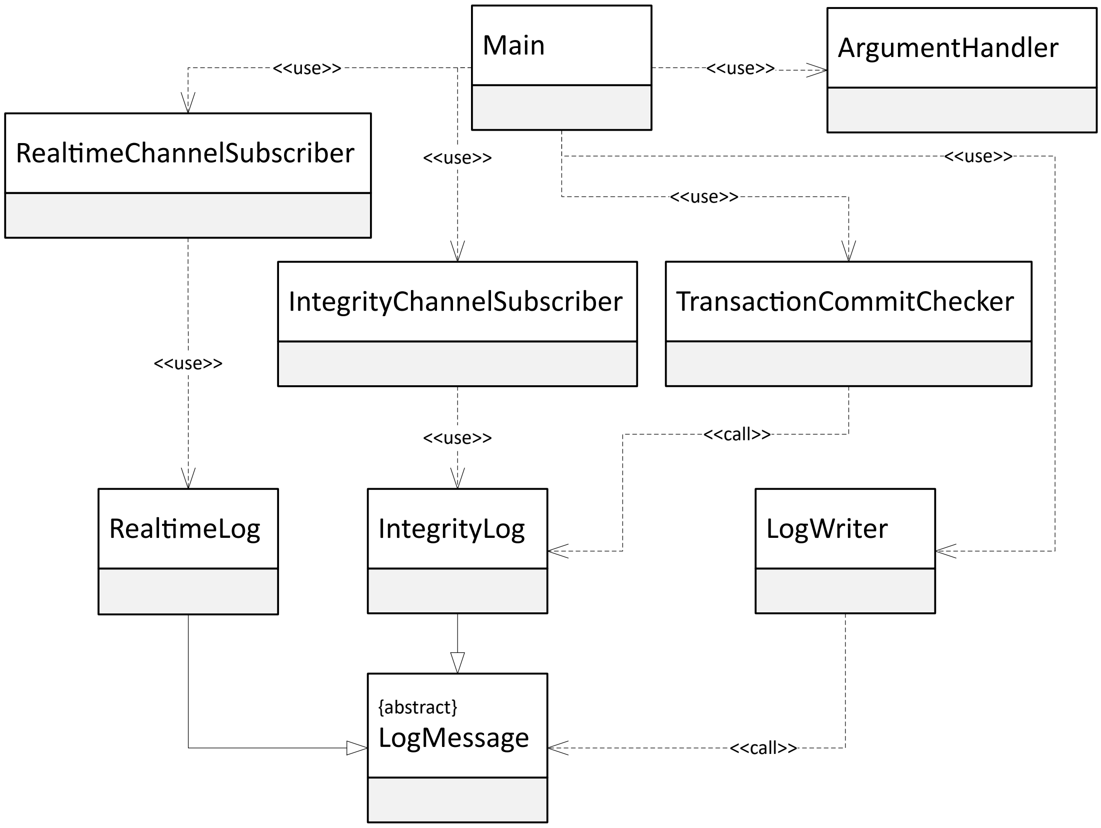


Figure 5.8: IoT Client Class Diagram

through the integrity channel. Since these channels operate independently, the values are also received independently. Thus, the two logs for the same value have to be matched. Both, *RealtimeLog* and *IntegrityLog*, contain a *Message* object of Figure 5.3. When these two *Message* objects are equal, this means that they contain the same measured value and hence they can be matched. Consequently, the matching *RealtimeLog* and *IntegrityLog* are written into the same line of the CSV file. This matching and writing is done for every *RealtimeLog* until none is left.

The *RealtimeChannelSubscriber* subscribes to the topics of the real-time channel and logs the incoming messages. The subscription is done via the defined MQTT broker. There are two options how the topics of the real-time channel can be defined. The first option is the initialization of a string array in the *Defines* class. This class contains a set of static final variables which are used to configure the IoT client. This string array can be passed to the *subscribe*-method of the *RealtimeChannelSubscriber*. The second option is to generate the topics automatically. This functionality is needed when operating with the virtual driver. Since the topics have always the same scheme, it can be generated without effort. The scheme looks as follows: "coap://<id>/<number>" which, for example, is "coap://192.168.0.10/0". Hence, the topics of the virtual driver are a sequence of strings with the number from 0 to $n-1$ whereas the number of data sources n depends on the number of active data sources in the virtual driver.

The *IntegrityChannelSubscriber* comprises two tasks. One task is the observation of new logs and the second task is the observation of new blocks generated in the blockchain. The observation of emitted events in the Ethereum network can be achieved with filters [99]. In Ethereum there are three classes of filters, namely block, pending transaction and topic filters. Block filters detect new blocks in the network whereas pending transaction filters detect new transactions. Topic filters allow specific filter criteria which can be defined. If the filters are used directly via the JSON-RPC API, continuous polling has to be done to obtain the updates for the filters. Furthermore, the block or transaction filters only return the according hash. If the full block or transaction is needed, another request has to be made. Web3j hides these extra steps by providing an asynchronous event-based API which does the polling and fetching of further information automatically. Since the polling is done by the Web3j instance, the instance has a default polling interval of 15 seconds. However, all timestamps which are recorded by the IoT client have an accuracy of milliseconds. Because of that, the Web3j instance is instantiated with a polling time of 1 millisecond. To observe the block generation, a block filter is created. In case a new block is detected by the filter, the timestamp is recorded. The recorded blocks with their timestamps are needed in the *TransactionCommitChecker*. For the observation of the emitted events of the smart contract, the method *ethLogObservable* is used which is actually a topic filter. To observe the smart contract, the wrapper class *IntegrityService* is not needed. The filter is only instantiated with the contract's address as well as the start and end block for the filter. Since only the latest logs shall be filtered, the start and the end block are parameterized with the predefined value *LATEST*. When an event is received, the first step is to decode it by reconstructing the IPFS hash. Thus, the first step is to look up the hash function type. With the hash function type and the received digest, a new multihash is generated. The hash is then used to look up the JSON file in the IPFS store. Since IPFS is tamper-resistant and every corruption of the data is detected [74], the IoT client does not have to do further validation procedures. Thus, computing the received data's hash and comparing it to the hash, which is stored in the blockchain, can be skipped. Only the existence of the file in IPFS proofs the integrity of the data. After the reception of the data, an *IntegrityLog* is generated.

The *TransactionCommitChecker* is used to get the timestamp when a specific block is confirmed by a specified amount of blocks. In case of this application, the amount of confirming blocks is eleven because Ethereum is used. A confirmation of eleven blocks in Ethereum means that the block containing the transaction of interest remains in the blockchain with a high probability [82]. With the collected *IntegrityLog* and the observed blocks of the *IntegrityChannelSubscriber*, the timestamps can be reconstructed. Algorithm 5.1 shows the procedure in the *TransactionCommitChecker*.

The result of the algorithm is that the timestamp of the eleventh confirmation is assigned to the specific log. The algorithm starts with iterating over all *IntegrityLog* elements of the input queue. The first step, in line 2, in the loop, is to check the removed status of the log. Every filtered log contains the “removed” tag [91]. This tag indicates whether the log is still in the main chain or not. The reason for a removal of the log is a chain reorganization. In case the log is already removed, the confirmation status has not to be checked since this log is not relevant anymore and the next element is checked. If the log is not removed, the algorithm continuous with the calculation of the block number for the eleventh block confirmation. Every block has a block number, which indicates the number of ancestor blocks [19]. Thus, to get the eleventh confirmation, eleven has to be added to the block number of the current block as it is done in line 3. After that, the blocks with the calculated block number are looked up in the recorded blocks. The search can have three different outcomes. First, in line 5, there is no block found which means that the block is not confirmed eleven times at this moment. The second case, in line 7, is that more than one block is found. This is possible because of the decentralization of the system and the concurrent generation of blocks by the different parties [19]. To find out which block is part of the main chain, the heaviest path must be found. This is the path where the most computation has been invested. The computation of a block can be seen from the difficulty in the block header. The difficulty is enough to validate the contributed computation [19]. Because of this, the total difficulty is used to validate the computation power. The highest total difficulty means the heaviest path, hence the block is part of the main chain. Equation 5.1 shows the definition of the total difficulty where B_t is the total difficulty, B_t' is the total difficulty of the parent block and B_d is the difficulty of the current block. When the block with the highest total difficulty is selected, then its timestamp is added to the log. The last case, in line 10, is that only

Algorithm 5.1: Set Transaction Confirmation Timestamp

```
   Input: Queue<IntegrityLog> Q, Map<EthBlock.Block, Long> B
   Output: Timestamps are set for elements of Queue Q
 1 for log in Q do
 2     if !log.isRemoved then
 3         n = log.blockNumber + 11
 4         commitBlocks = Get all blocks in B where blockNumber == n
 5         if commitBlocks.size == 0 then
 6             log.commitTimestamp = 0
 7         else if commitBlocks.size > 1 then
 8             b = Get block with highest totalDifficulty from commitBlocks
 9             log.commitTimestamp = B.get(b)
10         else
11             b = commitBlocks.get(0)
12             log.commitTimestamp = B.get(b)
13         end
14     end
15 end
```

Listing 5.6: IoT Client Start Command

```
USAGE: java −jar iotclient.jar −d <arg> −f <arg> [−i <arg>]
    [−t <arg>]
```

one block is found. In this case, the timestamp of this single block is set.

$$B_t \equiv B'_t + B_d \tag{5.1}$$

The *ArgumentHandler* class defines the options for the start of IoT client. Additionally, the options are parsed in this class and stored in a *CmdOptions* object. The start of the IoT client including the options is defined in Listing 5.6. The option d indicates, how long the client shall listen to the channels until the client stops listening and writes the CSV file. f determines the name of the CSV file for the report of the performance. These two options are required. The other two options are only required when virtual drivers are used. Since the addresses of the virtual drivers are generated, the two options indicate which topics of the real-time channel are available for the subscription. Thus, i specifies the id of the virtual driver and t the amount of topics which is available. With this information, the topics can be reproduced and the client is able to subscribe to them. In case i and t are not given, the topics are loaded from the *Defines* class.

The *Main* class is the entry point of the application. Here, the workflow of the application is determined. The first step is the parsing of the arguments with the *ArgumentHandler*. After that, the *RealtimeChannelSubscriber* is created and the connection to the MQTT broker is established. This explicit connecting in the *Main* class shall help to detect connection problems in the early stage of the application. In the next step the *IntegrityChannelSubscriber* is created and the observation is started. When both channels are listing to incoming events, the *Main* thread sleeps for a specified amount of time to collect enough data from the real-time and integrity channel. After that the observations are stopped and the *TransactionCommitChecker* is executed. Finally, all collected logs are written into a file with the *LogWriter*.

The content presented in the current chapter describes how the solution approach of Chapter 4 is implemented. The libraries, which are chosen for the implementation, enable a proper integration of the underlying technology in the applications. However, some documentations, especially the ones from the CoAP and MQTT libraries, are short. Thus, it takes a lot of time to find out some details of the libraries. Sometimes only code inspections can reveal the desired information. Another challenge is the testing, since the whole blockchain-IoT application has a lot of interfaces to different technologies. In case of the current implementation, the interfaces are tested manually. However, in bigger development projects automatic testing is essential to maintain a certain level of quality. Therefore, a lot of mock-ups have to be written to enable automatic testing and reach a higher test coverage. This additional effort has to be considered, when planning such software projects. Apart from the minimal documentation and the general testing issue, no further significant issues came up. An in-depth evaluation of the presented implementation shall give more information about the implementation's performance. The performance evaluation is part of the following chapter.

CHAPTER 6

Evaluation

This chapter contains the performance evaluation of the blockchain-IoT application on different IoT devices. As it has already been explained in Section 4.3, the blockchain-IoT application comprises the drivers, sensor or virtual driver, and the middleware. The first goal of this evaluation is to test the blockchain-IoT application and to create a proof-of-concept. The second goal is to test the performance on different devices. Thus, the portability of the blockchain-IoT application is tested and the IoT devices are tested for their suitability to run a blockchain enhanced application. To evaluate the blockchain-IoT application itself and the performance on the different IoT devices, several metrics are used. First, the message delays are measured on the real-time and integrity channel. This shall reveal the delay overloads which are caused by the integrity channel in comparison to the real-time channel. Furthermore, it is tested if the varying computing power of the devices has an impact on the delays. To evaluate the performance of the IoT devices, the resource usage, like CPU, memory and network traffic, is measured. Additionally, the CPU usage, which is consumed by every process, is measured. This shall identify anomalies in the usage of CPU of the different processes. Regarding the drivers, both drivers are used for the evaluation. The sensor driver is used to create a proof-of-concept with a real sensor. For the performance evaluation with different workloads, the virtual driver is used. By using the virtual driver, it is feasible to produce different workloads by varying the amount of data sources. For a systematic approach of the performance evaluation, the approach by Jain [100] is used. Jain defines ten steps which are common in all performance evaluation projects. Following these steps can avoid common mistakes. In the following part the performance evaluation is summarized in six steps.

1. System Definition: Since one goal is to test the suitability of different IoT devices for the blockchain-IoT application, three IoT devices are chosen. The devices are listed in Table 6.1. The constituted operating system is the system which is installed for the evaluation. Every device is set up individually, to meet the requirements of the underlying hardware. This is done by using images provided by the producers and in case required

Table 6.1: Used IoT Devices for the Performance Evaluation. (Source: [101, 102, 103])

	Raspberry Pi 3 Model B	Odroid-XU4	Intel Galileo Gen 2
Chipset	Broadcom BCM2837	Samsung Exynos5422	X1000
CPU	Quad Core @1.2GHz ARM Cortex-A53	Quad Core @2GHz ARM Cortex-A15 Quad Core @1.4GHz ARM Cortex-A7	Single Core @400MHz Intel Quark
Memory	1GB LPDDR2	2GB LPDDR3	256MB DDR3
Ethernet	10/100 MB/s	10/100/1000 MB/s	Available
Storage	MicroSD	MicroSD, eMMC 5.0	MicroSD
OS	Raspbian 8.0	Ubuntu 18.04	Yocto-built Linux

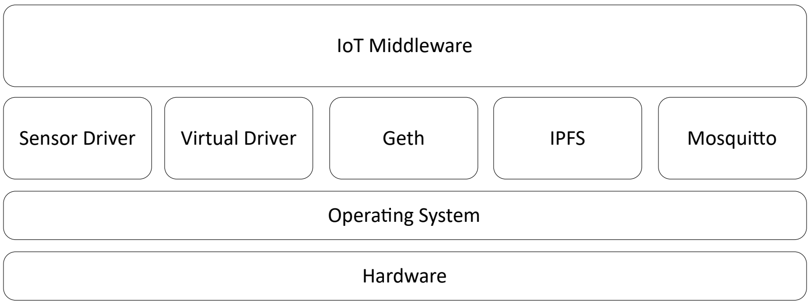


Figure 6.1: Setup of an IoT Device.

software is available in the operating system's repositories, it is installed from there. An advantage of this approach is that the heterogeneity of devices in IoT is reflected, hence the portability of the blockchain-IoT application can be tested. Additionally, using default software which is recommended by producers avoids potential issues with the hardware. Therefore, all operating systems which are installed on the devices are different, however, the operating systems are all based on Linux. Furthermore, the installed components like Geth, IPFS and Mosquitto are also installed individually, hence the versions of the components vary slightly between the devices. The version of Geth which is installed on the Raspberry Pi 3 is 1.8.8 and on the Odroid-XU4 1.8.9. The IPFS version 0.4.15 is on both devices the same. Regarding Mosquitto 1.3.4 is installed on the Raspberry Pi and 1.4.14 on the Odroid device. The installation of Geth and IPFS on the Intel Galileo is not possible, thus this device is not considered in the performance evaluation. A more detailed error report is given in Subsection 6.1.1. The whole setup of an IoT device is shown in Figure 6.1. This setup, comprising the IoT middleware and its components, is the first part of the evaluation setup. Regarding the driver only the Raspberry Pi supports the sensor and the virtual driver. The second part of the evaluation setup is the IoT client. The client is installed on a computer with 8GB memory, a Quad Core @3GHz processor and a 500GB *Solid State Drive* (SSD). Besides the IoT client, also

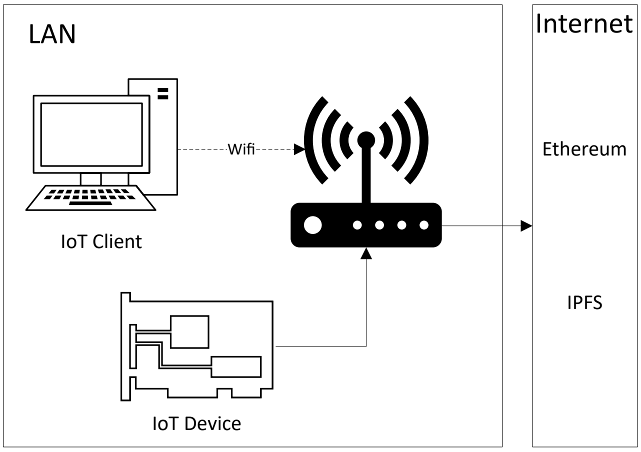


Figure 6.2: Connection of the Different Components in the Evaluation Setup.

Geth and IPFS are installed whereas Geth is not run in light client mode. Geth is run as a full node which means the whole blocks are synchronized not only the headers. The used Ethereum network is not the main network, instead the testnet Ropsten[1] is used.

As it has already been said, the first part of the evaluation setup is the IoT device with the installed components and the second part is the IoT client. To enable a communication between the two parts, they are connected to the same router. Thus, real-time channel messages and messages stored in IPFS are only exchanged in the *local area network* (LAN). Figure 6.2 sketches the whole evaluation setup. Since the delays of the network are calculated on timestamps which are taken on different devices, time synchronization is crucial. Hence, the *Network Time Protocol* (NTP) is used for the time synchronization. NTP is a protocol to synchronize clocks over the Internet [104]. The protocol is based on the IP and *User Datagram Protocol* (UDP). Since the timestamps which are taken by the driver and the IoT are accurate to the millisecond, the accuracy of the time synchronization has to be high. Measurements in a LAN where a router is between a client and a NTP server show that the average time difference is 1.6 μs with a standard deviation of 13.7μs [105]. Nevertheless, tests in the evaluation setup show that the offset is up to five milliseconds. This offset is acceptable, because the difference between real-time channel delays and integrity channel delays is much larger than five milliseconds. Therefore, the offset is sufficient and does not affect the delay analysis of the channels.

2. Services: The services which are provided by the blockchain-IoT application are the real-time and integrity channels. Messages which contain sensor data are sent via these two channels and clients can subscribe to these channels. On the real-time channel the client can subscribe to different topics which represent different data sources. On the integrity channel, a subscription to certain data sources is not provided, hence messages from all data sources are received. In the evaluation process the IoT client subscribes to

[1] https://ropsten.etherscan.io [Accessed: 2018-06-05]

both channels and to all topics of the real-time channel.

3. Metrics: Since one goal is the evaluation of the performance, metrics have to be chosen which cover the performance of the IoT devices and application. To evaluate the devices, their resource usages have to be measured. The evaluation of the different communication channels is based on the delays of the messages. The delay starts at the time when the value is measured in the driver and ends when the message is received at the IoT client. To achieve the evaluation goals, the following metrics are chosen:

- Total CPU Usage of IoT Device
- CPU Usage by Process of IoT Device
- Total Memory Usage
- Network Traffic of IoT Device
- Delay of Messages in Real-Time and Integrity Channel
- Confirmation Duration of Transactions

4. Parameters: The system parameters which affect the performance of the blockchain-IoT application are the following:

- Speed of the CPU
- Size of Memory
- Speed of the Network
- Speed of Ropsten Network

The workload parameters which affect the performance of the blockchain-IoT application are only the number of active sensors or rather the number of active data sources. The more data sources are connected to the middleware, the more data is sent and the higher is the workload.

5. Factors: The parameter which is varied during the evaluation is called factor whereas the different values of the factor are called levels [100]. During the performance evaluation different IoT devices are used. Hence, the parameter speed of the CPU and size of memory are factors whereas their levels are constituted in Table 6.1. Furthermore, the amount of data sources in the virtual driver is varied during the performance evaluation. Thus, the data sources are also a factor. It is important to mention, that the varying amount of data sources equals different workloads of sent data. However, the term data sources is used since the data loads, for the different experiments of the performance evaluation, are varied by varying the amount of data sources in the virtual driver.

To determine the levels of the factor data sources, it is first searched for benchmarks about common data loads of different IoT scenarios. Unfortunately, there do not exist benchmarks of similar performance evaluations. Thus, the different levels have to be determined differently. Since the regulation for monitoring and reporting greenhouse gas emissions of the European Union is used as motivational scenario, the idea is to derive the workloads for the experiments from this scenario. The report for greenhouse gas emissions of the European Union, which has to be produced annually, covers the emissions of seven greenhouse gases [106]. Furthermore, there also exist other report obligations, like the United Nations Framework Convention on Climate Change where eleven gases are monitored [107]. The distinct gases are phenomena. Since every phenomenon is managed by a data source, as it has been explained in Section 4.3.1, a data source is needed for every gas. Thus, in case of the European Union seven data sources are needed and in case of the United Nations eleven data sources are needed. These two cases are used as distinct workloads for the performance evaluation. Additionally, 22 data sources are chosen as third workload, to test the application and devices in extreme conditions. The fourth level is two data sources. The level two is chosen because on the Raspberry Pi a real sensor and consequently its driver are implemented. The sensor driver contains two data sources namely humidity and temperature. To have a proof-of-concept that the virtual driver is a good simulation of the real sensor driver, two is chosen as a level. To sum up, the different levels of the factor data sources are 2, 7, 11 and 22.

6. Experimental Design: The four levels of the factor data sources are executed on the Raspberry Pi and on the Odroid, which results in eight experiments. The ninth experiment is the execution of the physical sensor driver on the Raspberry Pi. All of the nine distinct experiments are performed three times to get an average of the metrics for the following analysis. Executing the nine experiments three times results in 27 runs. Every experiment has a duration of 60 minutes, which means the IoT client is listening for this amount of time on the two communication channels. This long duration shall give the possibility to track high delays especially on the integrity channel. Every experiment is executed as follows:

1. Start of Resource Usage Monitoring on the IoT Device

2. Start of the IoT Client

3. Start of the Middleware

4. Execute (a) or (b) Depending on IoT Device:

   a) Start of Virtual Driver and Send 60 Values per Data S

   b) Start of Sensor Driver and Run for 5 Minutes

6.1 Results

The results of the experiments are presented in this section. The analysis of the results focuses on two parts. The first part analyzes the delays of the real-time and integrity channels. The second part takes a closer look a the resource usage of the IoT devices. Section 6.1.1 presents the data which is collected during the execution of the experiments. Therefore, the data preparation and data losses are explained. On basis of the presented data, Section 6.1.2 analyzes the delays of the communication channels. Section 6.1.3 and Section 6.1.4 analyze the resource usage of the IoT devices.

6.1.1 Collected Data

The data basis for the analysis is collected during 27 runs of the specified experiments. This results in approximately 15,480 messages sent via the real-time channel and 15,480 log events emitted via the integrity channel. This is calculated by counting all data sources which are used in the 27 runs and multiply it with 60 which is the expected amount of values sent by a data source per run. Since not all runs are successful, data losses occur. As it has been said, it is expected that 60 values are distributed via the two communication channels for each data source in one run. In case fewer values are received, it is documented in Table 6.2. The column "Missing RT" shows the missing real-time channel values. It has to be mentioned that the Odroid device has a lot more value losses than the Raspberry Pi. Regarding the data losses of the integrity channel "Missing Int." both devices loose values only in case of 22 active data sources. It seems that the integrity channel is more stable than the real-time channel on the Odroid-XU4. During the execution of the different experiments, potential data losses are visible by several exceptions in the middleware which signalize issues with the IPFS and Mosquitto client. However, based on the thrown exceptions a single source or rather reason for the data losses can not be located.

Another anomaly in the data which is collected via the integrity channel is the negative

Table 6.2: Summary of Data Used for Analysis.

		Missing RT	Missing Int.	Neg. Confirmation	"Removed"	Used(%)
Raspberry	Phy	0	0	0	0	100
	2	0	0	0	0	100
	7	0	0	304	19	75.87
	11	0	0	184	5	90.71
	22	465	1325	319	5	80.08
Odroid	2	0	0	2	2	99.44
	7	156	0	84	0	93.33
	11	930	0	325	40	83.28
	22*	1316	466	113	29	56.14

* Only one run

confirmation time of transactions. Some subtractions of the transaction's reception timestamp and the eleventh block reception timestamp have a negative result. Since blocks in the blockchain are generated in sequence and have ascending block numbers, it is not possible that a block with a smaller block number has a timestamp which is bigger than the confirmation block's timestamp. A possible explanation for the negative confirmation time can be the different speeds of filters especially on a high rate of wanted incoming transactions. The IoT client uses two filters: One for the new generated blocks and the second one for the emitted events. Since the filters are executed in threads independently and the timestamps are taken when the event or block is found, the negative confirmation time can only occur when the event is found later than the eleventh confirmation of the block. A second anomaly, which may also be related to the same issue of an overloaded event filter, is the "Removed" status. Some transactions are stated as removed in the resulting data, which means that the events and therefore its transactions are removed from the blockchain. Although, random samples of these transactions are checked in the Ropsten[2] blockchain explorer and the explorer shows that the transactions are successfully committed. Thus, it can be assumed that on heavy loads of wanted incoming transactions, some transactions are overlooked by the filter. Therefore, the status of the last found transaction is kept, although the transaction which is part of the blockchain may already be received in a new block. All transactions with status "Removed" have a negative confirmation time because the IoT client does not set a confirmation time. Thus, the amount of negative confirmation times contain also the amount of "Remove" entries. The "Remove" entries are removed from the data set with the removal of the negative confirmation times. This preparation of the data set shall provide a clean data basis which can be used in further analyses.

The column "Used" shows how much of the expected data is used for the analysis of the delays. The expected data calculation is shown in Equation 6.1 where n is the number of registered data sources.

$$E = 3 * 60 * n \tag{6.1}$$

Equation 6.2 shows how the percentage of used data is calculated. The minimum data loss is taken from the real-time and integrity channels because the same value is sent via both channels. Taking the minimal data loss means how often the value itself got lost. As Table 6.2 shows, the experiment with 22 data sources is only executed once on the Odroid-XU4. The reason for that is that in the first run only four values are received via the real-time channel. Further runs showed similar results, hence run two and three are terminated early and no usable results are available. Therefore, this experiment is not only not suitable for a real-world use but it is also not suitable for a significant analysis. To calculate the "Used" value, the expected data is calculated for one run, not for three.

$$P = 100 * (E - min(MissingRT, MissingInt.) - Neg.Confirmation)/E \tag{6.2}$$

As it has already been mentioned, there is no data available for the Intel Galileo Gen2. The reason for that is that the processor does not support the *Multimedia Extension*

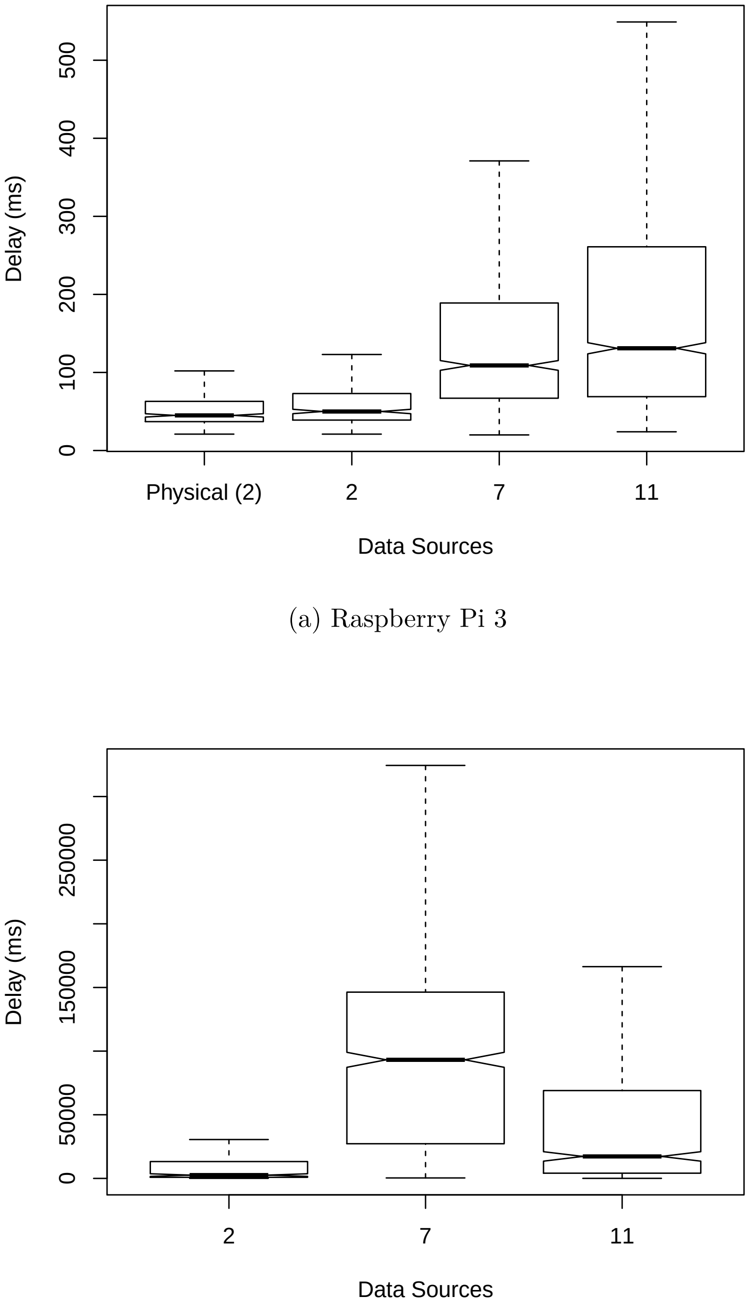
[2] https://ropsten.etherscan.io [Accessed: 2018-06-05]

(MMX) technology. Hence, Geth and IPFS are not running on the device. MMX is an Intel technlogy which increases the performance of CPUs and targets multimedia and communication applications [108]. The missing of the technology is indicated by the error message “This program can only be run on processors with MMX support” when starting Geth or IPFS. To check the validity of the error message, the supported features of the CPU are looked up with the command “cat /proc/cpuinfo”. The output confirms that the MMX technology is not supported by the CPU. Due to the failure of running Geth on this device, the Intel Galileo Gen2 is not suitable to run the blockchain in the IoT. Hence, it is not considered in the following performance evaluation.

6.1.2 Delay Analysis

In this section the delays of the real-time and integrity channels are analyzed. Besides statistical metrics, like median or quantil, notched boxplots are used to compare the data. The notch in the boxes indicates a 95% confidence interval [109]. Figure 6.3 shows the boxplots for the delays of the real-time channel of both devices. The boxplot of the 22 data sources experiment on the Raspberry Pi is missing because the median has a high increase. In case this would be plotted this distorts the scale. Thus, the 22 data sources experiment is missing to make the details and differences of the other experiments more visible. Regarding the Odroid device, there is not enough data available for a detailed analysis of the 22 data sources experiment.

Regarding Figure 6.3a, it can be said that the median of the delays is increasing with the amount of data sources. Furthermore, the box of the physical data source and the two data sources launched by the virtual driver achieve similar results, which shows that the virtual driver is a good simulation of the sensor driver. Another observation is that the upper whisker is increasing with the data sources. The lower whisker is almost stable. In comparison to the results of the Raspberry Pi, the Odroid has not a constant increase of the median which is constituted in Figure 6.3b. The experiment with seven data sources has the highest median, however, all medians are different at a confidence level of 95% because the notches are not overlapping. The huge gap between the first and the second box can be explained with the failed experiments for the configuration seven and eleven. The experiment with two data sources, where all runs are successful, has a similar result to the Raspberry Pi. Another observation with the failed experiments are the much higher medians. Table 6.3 shows the detailed values for the different statistical metrics. It reveals that also the successful experiment on the Odroid has a median of 2,613 milliseconds whereas the highest median of the Raspberry Pi’s successful experiment is only 321 milliseconds. Additionally, the variance is much smaller. Also the maximum delay of a successful experiment on the Raspberry Pi with 2,407 milliseconds is smaller than the Odroid median with two data sources. This is interesting because the Odroid is the stronger device in terms of the hardware, but the delays are much worse than on the Raspberry Pi. Another observation is that on failed experiments the delays are getting much worse, in comparison to successful experiments on the same device. The median on the Raspberry Pi nearly triples and the median on the Odroid is 35 times larger in



(a) Raspberry Pi 3

(b) Odroid-XU4

Figure 6.3: Real-Time Channel Delays.

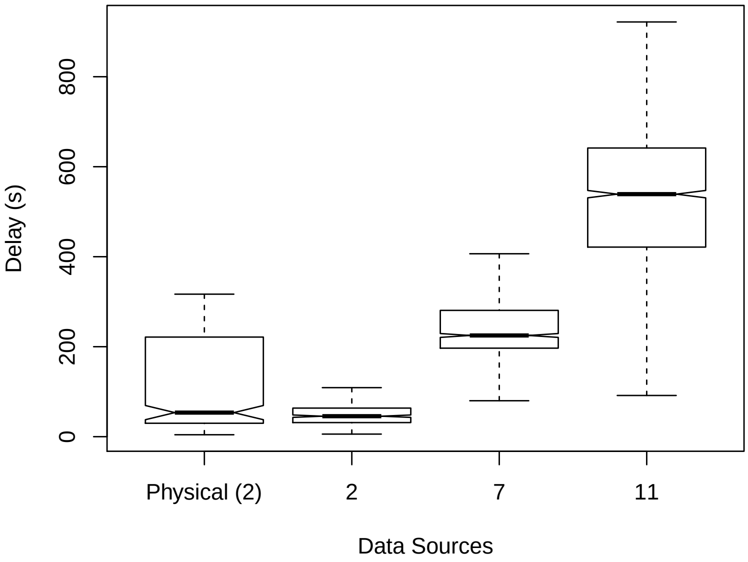
Table 6.3: Summary Delays (ms) Real-Time Channel.

		Min	Q1	Median	Mean	Q3	Max	σ
Raspberry	Phy	21.00	37.00	45.00	86.93	63.00	1785.00	169.234
	2	21.00	39.00	50.00	96.26	73.00	3238.00	258.605
	7	20.00	67.00	109.00	200.90	189.00	1856.00	268.4895
	11	24.00	69.00	131.00	253.80	261.00	2407.00	350.524
	22	26.00	129.00	321.00	2628.80	864.50	146121.00	15662.40
Odroid	2	54.00	872.80	2613.00	9411.10	13245.50	60104.00	13075.06
	7	356	272380	93165	103630	146321	352462	83819.53
	11	44	4103	17265	56806	69005	235261	73998
	22	-	-	-	-	-	-	-

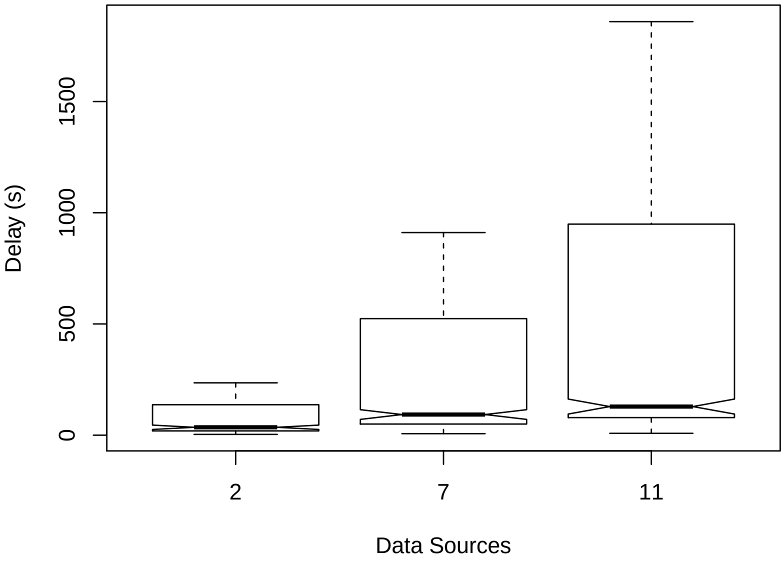
the worst case.

The second communication channel is the integrity channel. Figure 6.4 shows the delays of the integrity channel constituted as boxplots. The reasons for the missing boxplots of the 22 data sources experiment are the same as they are discussed for Figure 6.3. Similar to the real-time channel the median increases with the number of data sources. Also the median on the Odroid has a smooth increase. Nevertheless, the delays are much higher than on the real-time channel, which is an expected result. The higher range of delay values of the physical experiment attracts attention. However, the confidence intervals of the physical and the two data sources experiment are overlapping, which means that the median is not significantly different. The higher quantils may result from a slow Ropsten network, when a run was executed. Another observation on the Raspberry Pi is the strong increase of medians. It seems that the integrity channel scales not as well as the real-time channel on the Raspberry Pi. In comparison, the medians of the different experiments on the Odroid are increasing in smaller steps. However, the upper whiskers and the quantils are higher than the ones on the Raspberry Pi. In general it can be said that the variances on the Odroid are higher. Regarding Odroid's integrity channel, it seems that this channel is more stable than the real-time channel. In comparison to the real-time channel, value losses occur at first on the 22 data sources experiment. Table 6.4 shows the summary of the delay data in more detail. This table shows that the median of the physical experiment is only eight seconds higher than the median of the two data sources experiment. Comparing the medians of the Raspberry Pi and the Odroid, the latter is always faster except with two data sources. However, the Q3 and the maximum is better on the Raspberry Pi for 7, 11 and 22 data sources. Thus, it can not be said clearly which device achieved a better performance, since the high variances on the Odroid relativize the better medians.

Table 6.5 documents the confirmation delays for the received transactions. It can be seen that there is no trend in the confirmation times on increasing data sources. The median is more or less stable for each device. The variations may result from the current speed of the network and the performance of the filter. The median on the Odroid is



(a) Raspberry Pi 3



(b) Odroid-XU4

Figure 6.4: Integrity Channel Delays.

Table 6.4: Summary Delays (s) Integrity Channel.

		Min	Q1	Median	Mean	Q3	Max	σ
Raspberry	Phy	4.302	29.858	53.499	99.600	221.404	316.739	97.614
	2	5.739	31.448	45.538	51.359	63.503	143.668	29.289
	7	20.15	197.19	224.95	238.71	280.20	513.61	115.634
	11	52.11	422.03	539.03	526.52	641.58	1180.60	217.532
	22	47.45	723.11	992.98	1172.06	1503.20	3409.93	694.025
Odroid	2	3.461	19.130	76.36	118.44	143.75	423.88	68.567
	7	6.771	50.079	92.763	261.824	523.978	910.902	282.315
	11	8.259	78.938	127.727	472.022	943.437	1859.190	549.939
	22	65.37	636.30	963.88	943.51	1318.51	1533.45	457.973

Table 6.5: Summary Confirmation Delay (s) Integrity Channel.

		Min	Q1	Median	Mean	Q3	Max	σ
Raspberry	Phy	53.0	128.4	155.8	153.6	187.7	258.4	42.833
	2	54.92	91.10	129.57	147.43	195.40	315.12	68.671
	7	0.981	93.476	108.030	106.527	120.555	254.730	27.995
	11	1.654	85.072	130.343	132.617	164.869	265.441	56.536
	22	0.007	51.408	106.932	105.113	153.386	297.532	65.584
Odroid	2	21.95	63.74	76.36	118.44	143.75	423.88	87.940
	7	0.256	63.20	89.309	102.394	131.826	361.936	50.740
	11	1.792	36.694	94.289	91.485	135.890	245.124	61.034
	22	10.58	60.44	88.64	90.70	118.33	184.19	41.397

smaller than on the Raspberry Pi. All in all the confirmation times seem to be more or less stable and are not affected by the different loads of data sources.

To sum up, it can be said that the real-time channel is not working well on the Odroid in terms of delays and lost values. On the Raspberry Pi, it runs quite stable and scales well when the number of data sources increases. In comparison, the scaling on the integrity channel contains bigger gaps between the medians of the single experiments. In terms of the median, the integrity channel scales better on the Odroid, however the variances of the values are much higher than on the Raspberry Pi. Thus, it can not be clearly determined which device's performance is better regarding the integrity channel. Generally, it must be said that only the Raspberry Pi is worth to consider, because it has no value losses except for the 22 data sources experiment whereas the Odroid starts loosing values with seven data sources on the real-time channel.

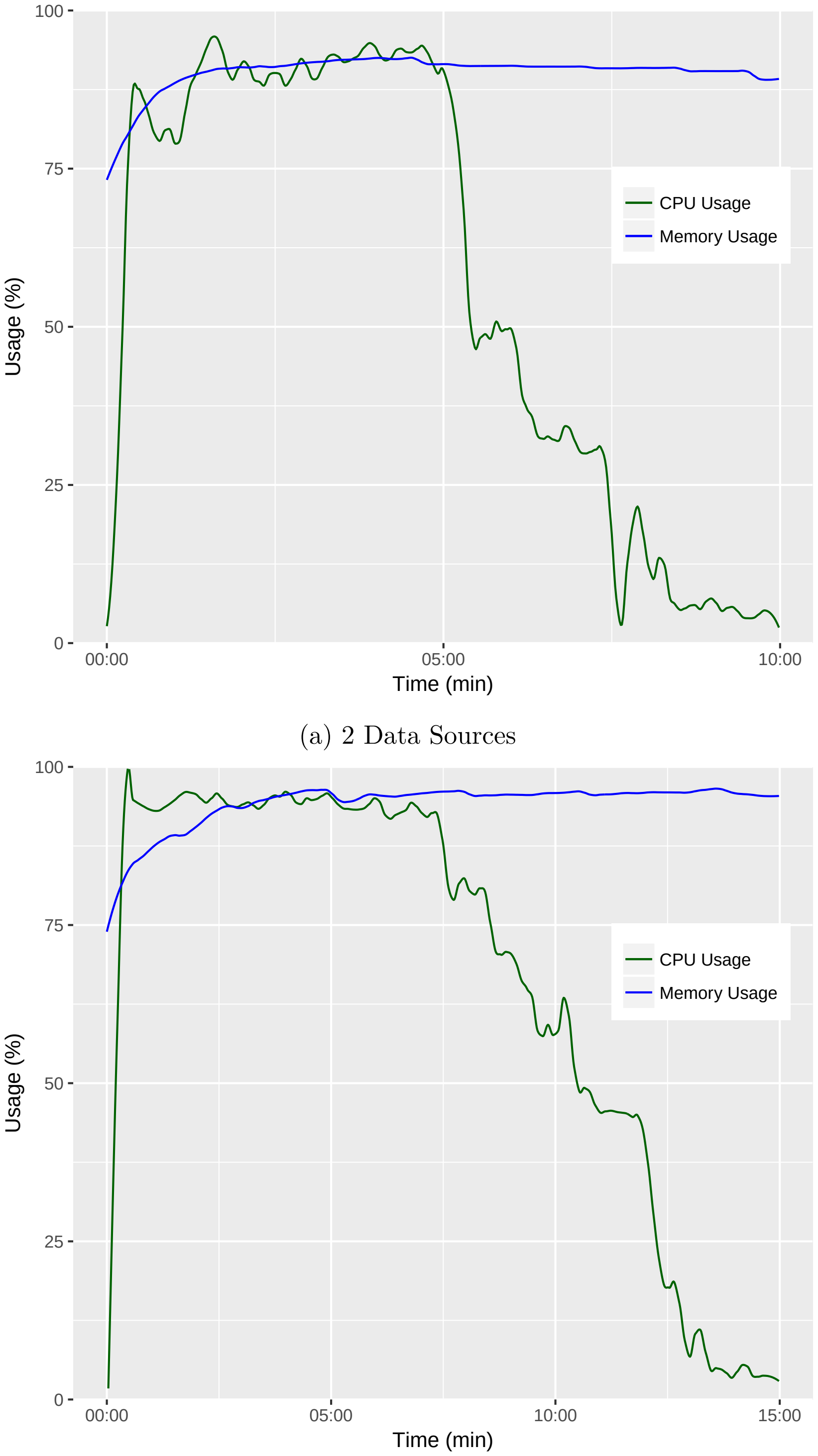
6.1.3 Total Resource Usage

This section deals with the analysis of the used resources during the experiments. To measure the performance, *nmon*[3] is used. nmon is a tool to monitor the performance of computers. The performance information can be directly seen on the screen or the information is saved to a file. In this performance evaluation, the performance information is saved to a file to use it for further analysis. The performance is measured every second. The resulting data, from all three runs, is plotted in a graph whereas *Locally Weighted Least Squares Regression* (LOESS) is used. LOESS is used to smooth scatter plots and to estimate trends [110]. Additionally, it is a robust method which means outliers have less impact on the result, which is in case of the plot the curve. The smoothing is done to make the trends in the data more visible and therefore to ease the analysis. The analysis is divided into two parts whereas the first part looks at the total resource usage in terms of CPU, memory and network traffic. The second part deals with the resource usage per process and is documented in Section 6.1.4. This shall reveal possible processes, which use too many resources and may cause the data losses due to the extensive resource usage.

Figure 6.5 shows the CPU and memory usage for two experiments executed on the Raspberry Pi. In both experiments no value losses occurred. Regarding the memory usage, it is pretty constant in both plots. In terms of the CPU usage, the course of the curve is similar. At the beginning there is a steep ascent, then the usage stays high for the five minutes where the driver is sending the values. After that, the usage decreases whereas with two data sources the decrease starts at minute five. In contrast, the usage descent with eleven data sources starts 2.5 minutes later. The median CPU usage is 92.75% with two data sources and 95.5% with eleven. Both have a maximum of 100%. Regarding the network traffic the median is 294.9 KB/s with two data sources and 320.8 KB/s with eleven. In comparison to the increase of data sources, the increase of data traffic is smaller. Figure 6.6 shows the resource usage of the failed experiment with 22 data sources. Regarding CPU usage and memory usage, no real anomalies are visible. The only difference is that the decrease of the CPU usage is slower than with eleven data sources. However, looking at the increase of the point where the descent of the CPU usage between two and eleven data sources starts, this seems to be a normal behavior. The median of CPU usage is 96.2% and the memory usage is nearly the same as with two data sources. The network traffic with a median of 255.2 KB/s is lower than with two data sources. The longer processing period and the loss of values can be the reason for the lower network traffic per second.

Figure 6.7 shows the CPU and memory usage of the Odroid. In Figure 6.7a the powerful hardware is visible. The median CPU usage is only 57.8% and the maximum never reaches 100%. In the experiment with eleven data sources, where value losses occurred, the median is 88.9% which is a higher increase of the usage from two to eleven data sources than on the Raspberry Pi. In this experiment CPU usage rates of 100% are

[3] http://nmon.sourceforge.net/pmwiki.php [Accessed: 2018-07-25]



(a) 2 Data Sources

(b) 11 Data Sources

Figure 6.5: CPU and Memory Usage on Raspberry Pi 3.

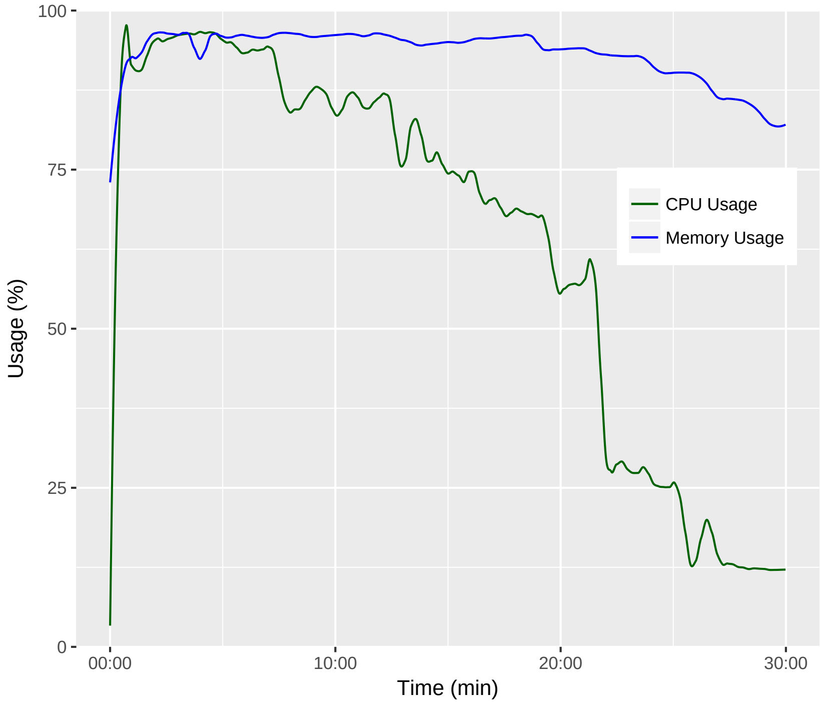
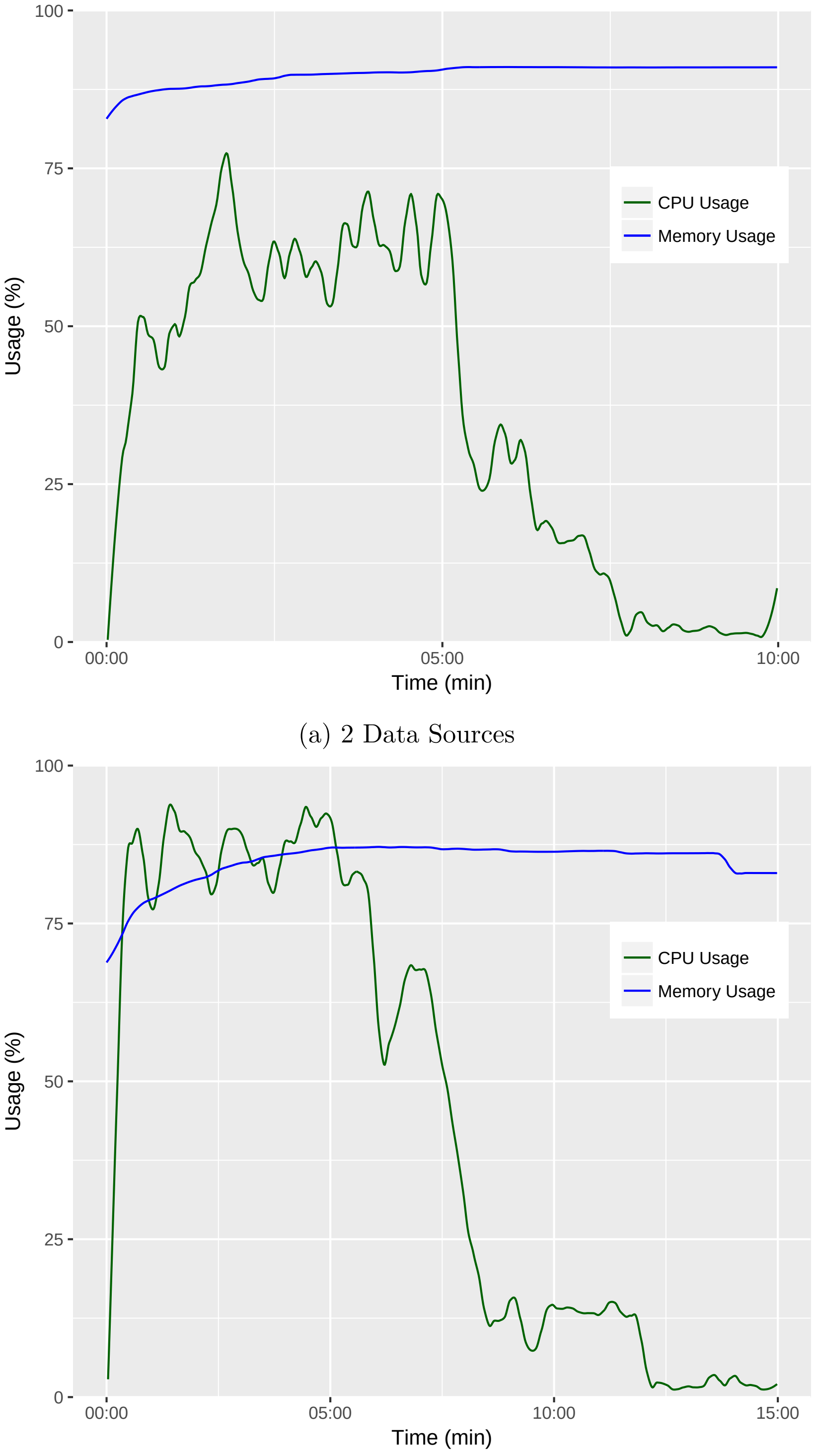


Figure 6.6: Experiment with 22 Data Sources on Raspberry Pi 3.

reached. The memory usage is slightly higher in the two data sources experiment than on the eleven data sources experiment. Comparing the course of the usage between the Raspberry Pi and the Odroid, it can be said that the course of the usage is similar in terms of the CPU and the memory. The difference is that the Odroid has a lower usage and the descent after five minutes is faster. The network traffic has a median of 360.9 KB/s for two data sources and 489.9 KB/s for eleven. Compared to the Raspberry Pi, the network traffic is higher for the same sent data.

Table 6.6 summarizes the results of the resource usage. It is obvious that the Odroid has a lower CPU usage in all experiments. Although, the *Median Absolute Deviation* (MAD) on the Odroid is higher than on the Raspberry Pi especially on the two data sources experiment. In all experiments a maximum of 100% is reached expect, on the two data sources experiment on the Odroid where only 95.70% are reached. The CPU usage is high on the Raspberry Pi, but the MAD is quite low with a maximum of 6%. An interesting aspect of the CPU usage is that the Raspberry Pi has a higher usage, however, it looses fewer messages. Furthermore, between the eleven and 22 data sources experiment on the Raspberry Pi there is no big difference between the CPU usage, however, in the 22 data sources experiment value losses occur. It seems that a certain threshold is passed and value losses start to occur in both communication channels. Regarding the memory, the Raspberry Pi has nearly the same usage as the Odroid and on the two data sources experiment the usage is slightly lower. The interesting thing about it is that the Odroid



(a) 2 Data Sources

(b) 11 Data Sources

Figure 6.7: CPU and Memory Usage on Odroid-XU4.

Table 6.6: Total Resource Usage.

		CPU (%)				Memory (%)			
		Median	MAD	Q3	Max	Median	MAD	Q3	Max
Rasp.	2	92.75	6.00	95.80	100.00	93.30	4.69	96.08	97.59
	11	95.50	3.26	97.20	100.00	94.49	2.76	96.25	96.82
	22	96.20	3.41	98.10	100.00	96.27	1.08	96.59	97.73
Odr.	2	57.80	21.94	71.60	95.70	96.03	3.01	97.68	98.52
	11	88.90	9.64	94.20	100.00	89.90	11.42	96.75	97.94

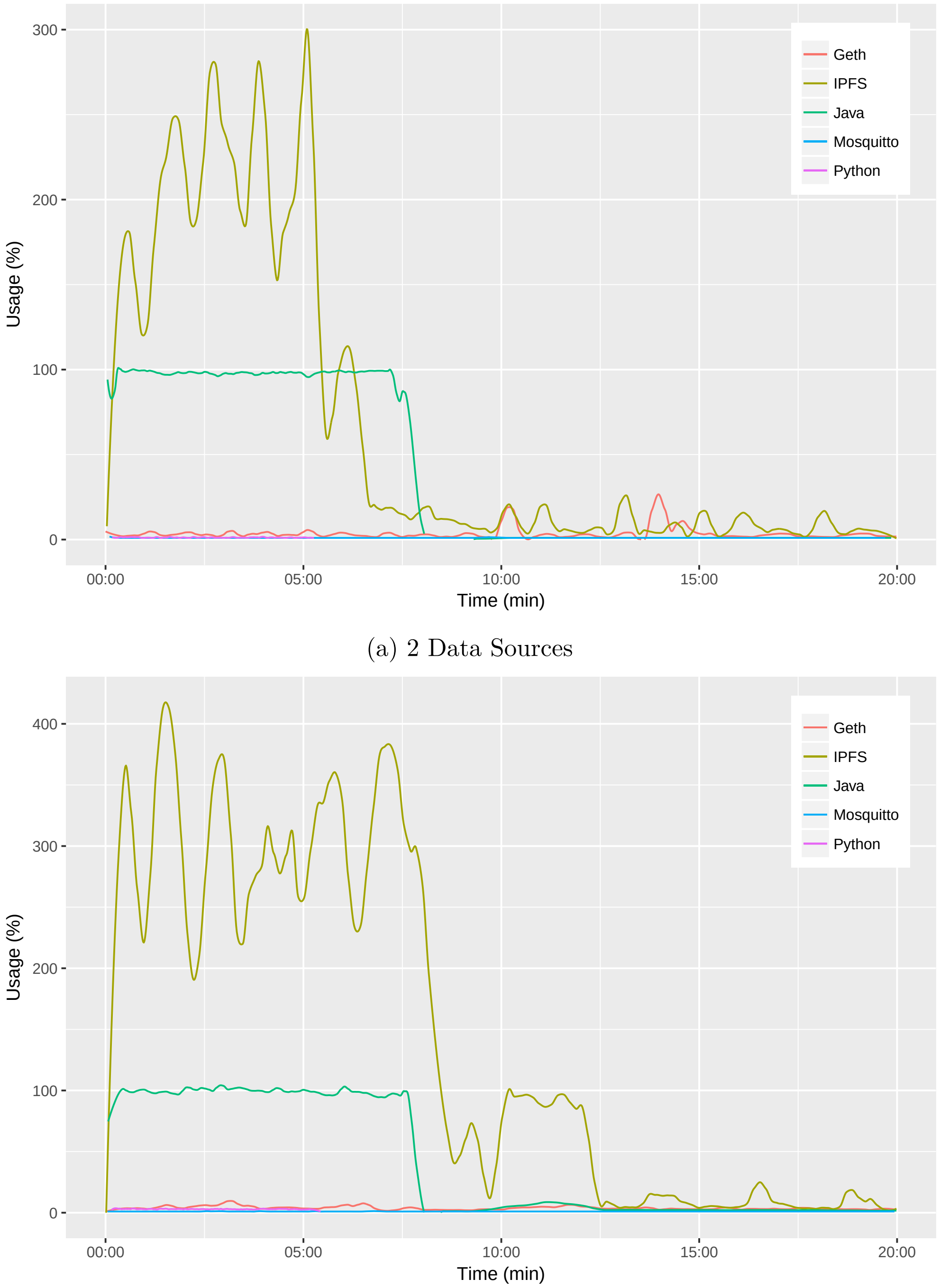
has twice the size of the Raspberry Pi's memory, but the relative usage is still higher. Regarding the MADs of the memory, they are all quite low. The only value which is clearly higher is the MAD of the eleven data sources experiment on the Odroid.

6.1.4 Resource Usage by Process

This section looks at the CPU usage per process whereas the processes Geth, IPFS, Java, Mosquitto and Python are considered. Basically, the measurements for this processes are also done every second, however, in the resulting data set the measured timestamps are quite irregular. The reason for that is that the processes are only documented when they use a significant amount of CPU during the specified interval. Furthermore, in the documentation of the *nmon analyser*[4] it is explained that the CPU usage can be greater than 100% because nmon shows the usage of a single CPU, not the usage of the whole system. Thus, it is possible that multi-threaded processes consume more than 100%.

Figure 6.8 shows the results for the Odroid. The first thing to be noticed is the by far highest CPU usage of the IPFS process with an approximate maximum of 550% in both experiments. However, the median of the eleven data sources experiment is 341.77% and with two data sources nearly the half is used namely 170.71%. The second highest usage is done by the Java process, which executes the middleware and has an approximate median of 99%. All other processes do not have a significant usage. The same behavior of the processes can be observed on the Raspberry Pi. Figure 6.9 shows the usage of the processes whereas IPFS has also by far the highest CPU usage. On the failed experiment with 22 data sources (Figure 6.10b) the processing period is very long with a maximum usage of 520.58% only in the IPFS process. In comparison, the maximum usage with eleven data sources is 398.98%. The median in the failed experiment is 241.21% and in the eleven data sources experiment 262.47%.

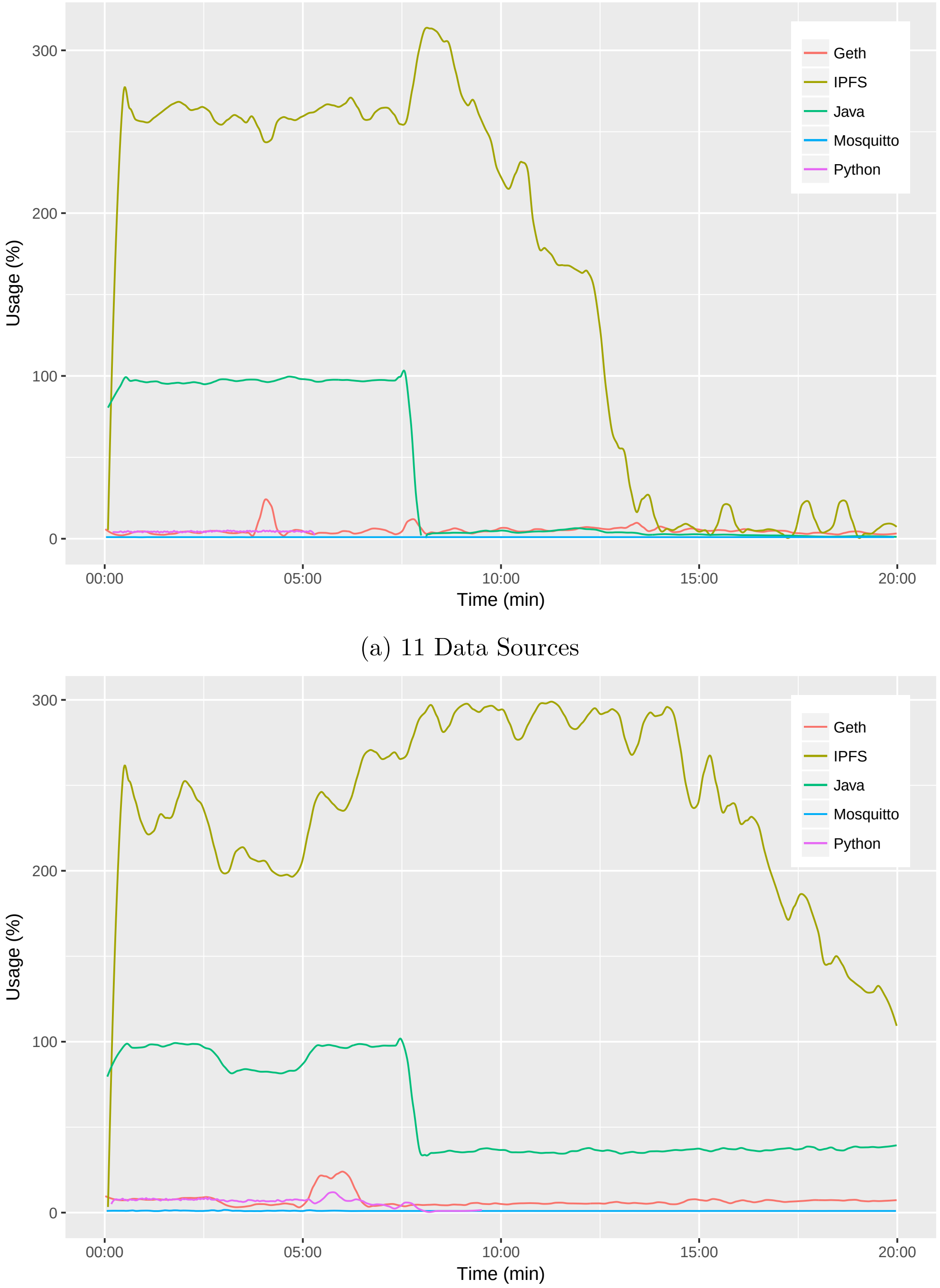
[4] https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Power%20Systems/page/nmon_analyser [Accessed: 2018-08-04]



(a) 2 Data Sources

(b) 11 Data Sources

Figure 6.8: CPU Usage by Process on Odroid-XU4.



(a) 11 Data Sources

(b) 22 Data Sources

Figure 6.9: CPU Usage by Process on Raspberry Pi 3.

Table 6.7: Delays with Only Real-Time Channel (22 Data Sources).

Min	Q1	Median	Mean	Q3	Max	σ
Raspberrry Pi 3						
20.0	53.0	92.0	213.7	209.0	2331.0	314.5
Odroid-XU4						
33.0	543.0	672.0	989.8	882.0	9938.0	946.6

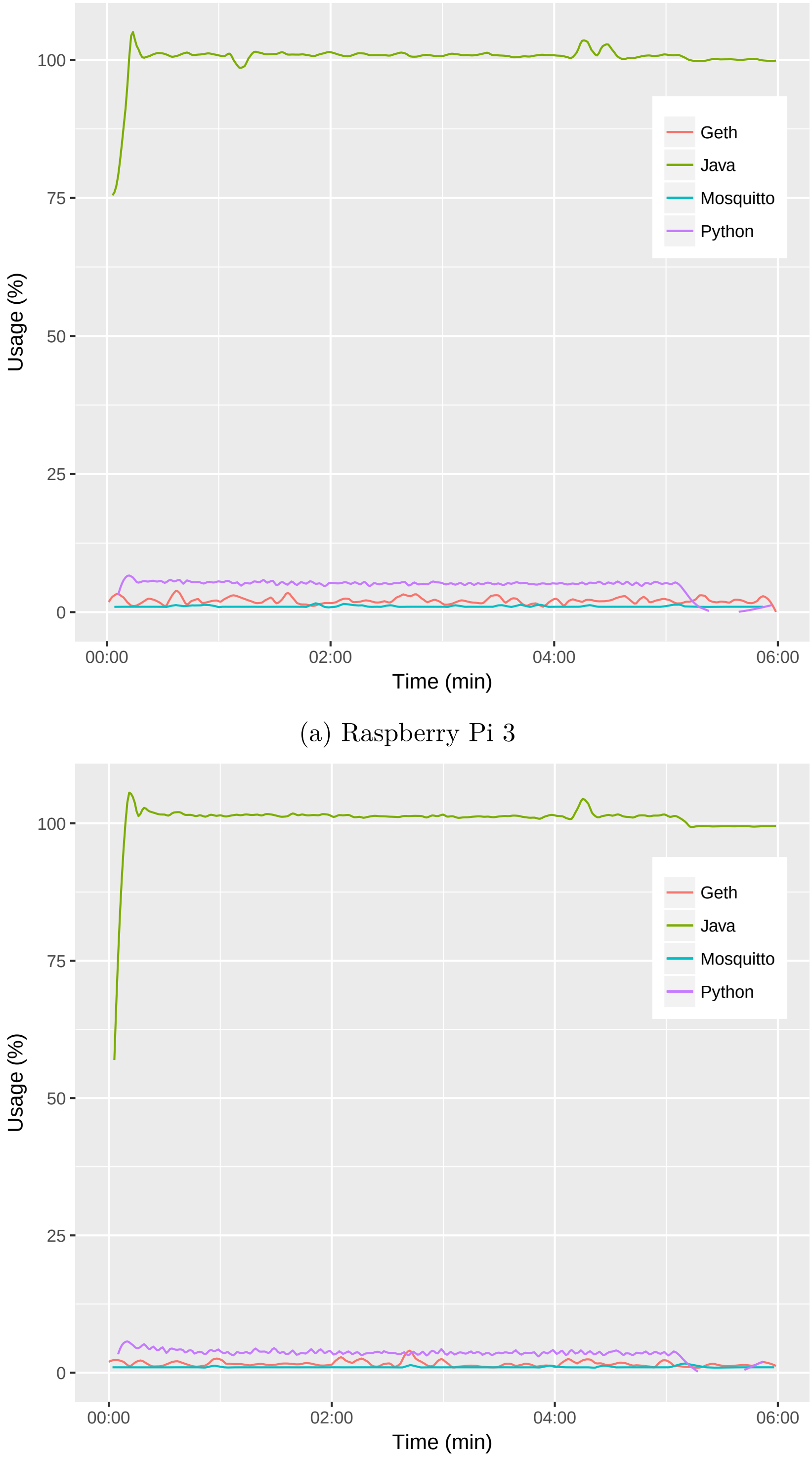
6.1.5 Only Real-Time Channel

Since the results in Section 6.1.4 reveal that IPFS consumes a lot of resources, the evaluation procedure, which is explained at the beginning of this chapter, is slightly changed for another experiment. Therefore, an experiment is designed which is executed with 22 data sources and the integrity channel of the middleware is switched off. The only channel where data is sent, is the real-time channel. Additionally, the IPFS client is shut down. The Geth client is running and synchronizing, however, no transactions are issued because the integrity channel is switched off. The experiment is executed three times on the Raspberry Pi 3 and the Odroid-XU4.

The first result which has to be mentioned is that no value losses occur on both devices, although, value losses occur in the experiment with both communication channels switched on. Furthermore Table 6.7 shows, the median delay on the Raspberry Pi is only one third and the Q3 is only a quarter in comparison to original experiment (Table 6.3). Regarding the Odroid results, there is no comparison to the original 22 data sources experiment, however, the delay is even better than the two data sources experiment. It can be said that the integrity channel has a huge impact on the performance of the real-time channel. Since the Geth client is still synchronizing during the experiments, it seems that IPFS causes the problems. Despite the improvement of the delays, the Odroid has still longer delays than the Raspberry Pi.

Table 6.8 shows the total CPU and memory usage of the experiments. The more powerful hardware of the Odroid is visible. Compared to the experiment with both channels activated, only a minimal amount of CPU is consumed. The CPU usage of the Raspberry Pi is twice as much as on the Odroid. Regarding the memory, the usage is almost the same, which is interesting, because as it has already been said, the Odroid has twice the size of memory than the Raspberry Pi. For a more detailed look at the resource usage, Figure 6.10 shows the resource usage per process. However, compared to the figures of Section 6.1.4 there are no clear differences, despite the IPFS process is missing.

To sum up, it can be said that the integrity channel has an impact on the performance of the real-time channel, in terms of making the real-time channel unstable, regarding the value losses, and the increasing of the delay times. Furthermore, it seems that the main troublemaker is IPFS. A reasonable explanation for the worse performance of the Odroid compared to the Raspberry Pi can not be found on basis of the shown results.



(a) Raspberry Pi 3

(b) Odroid-XU4

Figure 6.10: CPU Usage by Process with Only Real-Time Channel (22 Data Sources).

Table 6.8: Total Resource Usage with Only Real-Time Channel (22 Data Sources).

	CPU (%)				Memory (%)			
Device	Median	MAD	Q3	Max	Median	MAD	Q3	Max
Rasp.	26.40	0.89	28.60	70.20	66.09	2.25	67.19	68.48
Odr.	13.5	0.74	21.20	33.50	66.29	0.13	66.40	66.52

6.2 Discussion

Stability

It can be said that the integrity channel is more stable in terms of value losses. On the Odroid the value losses in the real-time channel start already with seven data sources whereas the value losses on the Raspberry Pi occur with 22 data sources. Determining the cause for the instability of the real-time channel on the Odroid is difficult for several reasons. At first the thrown exceptions in the middleware, when values get lost, are diverse and they are not the same in every error case. The exceptions are thrown by IPFS and the MQTT client. The second difficulty to trace down the failing of the real-time channel is the performance usage. No clear patterns can be recognized how the performance is different in case of data losses. The only anomaly in the performance data is the high CPU usage of the IPFS process. However, the pattern is not clearly different between an error case and a successful case. Regarding the maximum IPFS CPU usage on the Odroid, it is nearly the same in the successful and the failed experiment. On the Raspberry Pi the CPU usage, concerning the median, of the failed experiment, is 20% lower than the successful one, however, the maximum is 120% higher. Regarding these values, a clear difference in the performance is not obvious. Nevertheless, the experiment with the integrity channel shut down shows that the performance of the real-time channel is affected by the integrity channel. To sum up, it can be said that the integrity channel has fewer data losses when both communication channels are activated. This can be the preferred option for some use cases where integrity is more important than real-time updates. Thus, in terms of data losses the integrity channel is more stable.

Portability

Portability is a specified nonfunctional requirement for the blockchain-IoT application. The installation and execution of the middleware, the driver and the needed components is feasible. However, the performance evaluation shows that the devices' results are quite different. Interestingly, the device with the more powerful hardware achieves worse results in terms of number of delivered messages and delays in the real-time channel. The only experiment which runs without data losses on the Odroid is the one with two data sources. On the Raspberry Pi data is lost only with 22 data sources. Also the delays of the real-time channel are smaller on the Raspberry Pi. When the delays of the integrity channel are concerned, the Odroid is slightly better regarding the medians. In terms of portability, these results show that the portability is not working well. Despite the usage of platform independent languages and components, portability is still not achieved. This means that the development process should include an extensive testing phase on different devices, since relying on platform independent technology does only work to a certain degree.

Technologies

The blockchain-IoT application depends on a lot of different technologies. Furthermore, there are some of the technologies, like Geth or IPFS, which are not well established yet. Thus, the robustness of the technologies may not have reached a certain level of maturity and experiences are rather rare, especially in the IoT environment. However, the light client synchronization mode of Geth seems to work well on resource-restricted devices. In contrast, IPFS consumes a lot of the CPU resources even though a profile *lowpower* for resource savings is added in version 4.15. Due to the results of the performance evaluation, it can be said that IPFS is not a good option to be used on resource-restricted devices. Especially when the device is used not only as a IPFS node. The different technologies which have different levels of maturity make it difficult to trace occurring errors, since dependencies or impacts between the technologies are hard to discover. The issue has already been explained for the data loss problem. To gain more certainty about the used technologies, the technologies should be tested isolated on various devices. A final evaluation shall then reveal the suitability of the technology, to be used in the planned application. Another option is to concentrate on one experimental technology and use well-established ones for the rest of the application. In case of the blockchain-IoT application, the first try to stabilize the communication and decrease the CPU usage could be the exchange of the IPFS technology with another well-established database.

CHAPTER 7

Conclusion

The blockchain and the IoT are emerging technologies [1]. In general, it can be said that these two technologies are aiming for different application fields. Nevertheless, combining these two technologies has the potential to solve some challenges in the IoT [2, 3] and provides opportunities. Challenges in the IoT, like robustness and scalability, shall be improved by the blockchain's decentralized architecture [2, 3]. An opportunity for the IoT is the monetary exchange of data and computing power [76]. However, besides the opportunities and the potential solutions, there are also a lot of challenges to overcome when using the blockchain in the IoT. For instance, Blockchain technologies need a lot of resources [76]. Thus, running the blockchain on resource-restricted devices is a challenge. Another challenge are latency demands of time-sensitive applications which are not realizable with the current blockchain technologies, when the applications depend on fully confirmed transactions [76].

One contribution of this thesis is the development of the blockchain-IoT application which is able to collect data from sensors and distribute data via two communication channels. One channel distributes data in real-time, the other channel distributes data with guaranteed integrity by using Ethereum. Besides the blockchain-IoT application, an IoT client is implemented which monitors the delays of both channels and keeps track of transactions' confirmation times. Finally, the blockchain-IoT application is tested on different IoT devices whereas delays and resource usage is considered in the evaluation.

To get an impression about the state-of-the-art in terms of approaches combining the blockchain and the IoT, a survey of the related work is conducted in Chapter 3. Existing approaches are identified like the approach from Huh et al. [45], which sets policies for IoT devices on Ethereum via smart contracts. Another approach is the blockchain-based data integrity framework for IoT data from Liu et al. [47]. The impression from this state-of-the-art research is that a lot of papers deal with the topic blockchain and IoT on a low level. With low level it is meant that the papers focus on basic questions instead of integrating the technologies in more complex applications. Based on the research results, regarding the challenges, opportunities and current solution approaches, four research questions are formulated in Chapter 4. The research questions and their answers are part of the following section.

7.1 Discussion of Research Questions

As it has already been said, the four research questions, which are formulated in Chapter 4, are based on the research results from Chapters 2, 3 and 4. The research questions help to develop a better understanding of the software requirements when designing an IoT application which integrates the blockchain. In the following paragraphs, the research questions are answered, based on the results of this thesis.

RQ-1 How can the integration of the blockchain in the IoT be realized from a software engineering perspective?

To answer this question, an IoT application is developed which uses the blockchain. Chapter 4 describes the design of the application and Chapter 5 the actual implementation. The application uses the tamper-proof property of the blockchain to distribute sensor data with guaranteed integrity. The whole application comprises several parts. First, there is the sensor driver, which collects data from sensors and provides data via a CoAP interface. Second, the middleware collects the data from the sensors and provides it through two channels. One provides the data in real-time, the second distributes the data with guaranteed integrity. The integrity channel uses the Ethereum blockchain to ensure the integrity.

RQ-2 What is needed to suit time-sensitive applications and take advantage of blockchain properties?

Since current blockchains are not suitable for time-sensitive applications [76], the solution approach in Chapter 4 and the implementation in Chapter 5 do not try to realize a real-time distribution of data with the blockchain. Therefore, an extra channel is provided by the middleware which focuses on the time-sensitive delivery of the message. To distribute messages in real-time, MQTT is used. Besides the time-sensitive delivery of data, MQTT has another advantage. Clients are able to receive data from specific data sources, by choosing the according topic. Choosing the data by its source is not possible in the integrity channel.

RQ-3 How to store data while keeping the stored data volume in the blockchain low?

The approach to keep the data which is stored in the blockchain as small as possible is introduced in Chapter 4 and implemented in Chapter 5. The chosen approach in this thesis is to store the data off-chain and to only store the data's hash in the blockchain. This strategy keeps the data in the blockchain small and still guarantees the integrity of the data. As storage technology IPFS is used. An advantage of the technology is the P2P architecture and that nodes in the network do not have to trust each other [74]. Another advantage is that the identification of the data is done via hashes. These hashes can be stored in the blockchain. Despite the promising features of IPFS, the performance evaluation in Chapter 6 shows that this technology consumes by far the most CPU power. This high CPU usage does not suit well into the IoT environment.

RQ-4 How does a blockchain-IoT application perform on different devices?

In Chapter 6, the answers to this question are presented. The developed blockchain-IoT application is tested on three different devices, namely on an Intel Galileo Gen2, an Odroid-XU4 and a Raspberry Pi 3. Regarding the Intel Galileo Gen2 device, the blockchain-IoT application can not be executed, because the CPU does not support the needed MMX to run the Geth and IPFS client. Thus, the Intel Galileo is not an appropriate device to integrate the Ethereum Blockchain into the IoT. Regarding the Odroid-XU4, the setup of the application works without any issues. However, on increasing data sources the real-time channel looses a lot of messages. Also the delays of this channel are bigger than the real-time delays on the Raspberry Pi. The message losses on the integrity channel occurred only with 22 data sources. Regarding the Raspberry Pi 3, the message losses for both channels only occurred on the experiment with 22 data sources. It can be said, that the Raspberry Pi 3 performs the best in terms of message losses. In terms of delays on the real-time channel the Raspberry Pi also achieves better results, however, based on the delay data of the integrity channel a clear favorite can not be identified.

7.2 Future Work

Combining the blockchain with the IoT is a very extensive topic where a lot of different technologies are involved. Furthermore, it is also a rather new topic. Thus, there are still research questions which are not answered yet. In the course of this thesis further topics came up, which are interesting to be regarded in future works. The topics which are interesting to be considered in further explorations are discussed in this section.

Alternative Storage

The IPFS process needs, on both tested devices, by far the most CPU power in comparison to the other processes, as it is shown in Section 6.1.4. Therefore, an alternative is needed to lower the resource usage. Different strategies can be imagined. The first idea is to change to a classical database. However, a challenge is the centralization, which creates a potential bottleneck. Thus, distributed approaches have to be considered and tested. In case IPFS is still considered as storage technology, the research has to focus on the availability of the data which is stored locally in the IoT device. Therefore, cluster approaches could be explored.

Exception Handling

As it has already been said several times, combining the blockchain and the IoT comprises many technologies, interfaces and components. Furthermore, the IoT is a rough environment where applications have to deal with resource restrictions, lossy links and heterogeneous devices. Also the blockchain is an environment which needs a flexible handling. For instance, the speed of the transaction delays with partly large outliers in the integrity channel show, how flexible applications have to be to react to these behaviors. The handling of lossy links and the optimizations of technologies for the IoT are already included in technologies like CoAP. However, the exception handling for the processes in the blockchain and the development of best-practice methods need further research effort.

Bidirectional Communication

At the moment the communication is only unidirectional, in terms that the data is sent from the IoT device to the client. However, besides sensors there also exist actuators which can be connected to IoT devices. To be able to send commands to the actuators, the middleware must also support the reception of data. The integrity channel can be used for logging critical commands which are sent to the actuators.

Automatic Deployment

The distribution of sensor data, whereas the integrity of the data can be guaranteed, is one central requirement of the implemented blockchain-IoT application. This is realized by using the blockchain. An additional advantage by using the blockchain is the transparency of the distribution process. However, the application itself and its deployment is not transparent. Thus, an interesting question is how to deploy the application in such a way that deployment is more transparent and the integrity of the application itself can be guaranteed. To take the deployment process a step further, an autonomous deployment of the application, which is decided by IoT devices, is imaginable. This would enable an autonomous sensor network to decide where the collection of sensor data is needed. In case a suitable device is found, the application is deployed to it and the communication channels are established. However, if this is realized the transparency of the deployment process is crucial.

List of Figures

List of Tables

List of Algorithms

Listings

Acronyms

AMQP *Advanced Message Queuing Protocol.* 49

API *Application Programming Interface.* 16, 20, 61, 64, 68

ASIC *application-specific integrated circuit.* 12, 30

CoAP *Constrained Application Protocol.* 34, 35, 43, 49–53, 57, 58, 70, 98, 100

CPU *central processing unit.* 4, 73, 74, 76, 80, 85–95, 99, 101

CSS *Cloud Storage Service.* 24

CSV *Comma Separated Values.* 66, 67, 70

DAC *decentralized autonomous corporation.* 26

DAG *directed acyclic graph.* 32, 38

DCAs *Data Consumer Applications.* 24

DDS *Data Distribution Service.* 34, 35, 49

DHT *distributed hash table.* 28, 35

DIS *Data Integrity Service.* 24

DOA *Data Owners Application.* 24

EVM *Ethereum Virtual Machine.* 12, 30, 64

GHOST *Greedy Heaviest Observed Subtree.* 30, 31

HTTP *Hypertext Transfer Protocol.* 60–62, 64

IoT *Internet of Things.* xi, 1–5, 7, 8, 15–21, 23–31, 34, 35, 37–41, 43, 44, 46–51, 54–56, 65–68, 70, 73–80, 94, 95, 97–101, 103

IP *Internet Protocol.* 51, 53, 75

IPC *inter-process communication.* 61

IPFS *InterPlanetary File System.* 35, 49, 50, 57, 60, 62, 64, 65, 68, 74, 75, 78, 80, 89, 92, 94, 95, 98, 99

IPv4 *Internet Protocol Version 4.* 51

IPv6 *Internet Protocol Version 6.* 51

JSON *JavaScript Object Notation.* 49, 53, 54, 56, 58, 61, 65, 68, 105

LAN *local area network.* 75

LES *Light Ethereum Subprotocol.* 60

LOESS *Locally Weighted Least Squares Regression.* 85

M2M *machine-to-machine.* 18, 31, 58

MAD *Median Absolute Deviation.* 87, 89

MMX *Multimedia Extension.* 79, 80, 99

MQTT *Message Queuing Telemetry Transport.* 34, 35, 49, 50, 57, 59, 67, 70, 94, 98

NFC *near field communication.* 18

NTP *Network Time Protocol.* 75

P2P *peer-to-peer.* xi, 2, 9, 13, 25, 28, 35, 38, 39, 50, 64, 99

Perm *permissioned.* 32, 33

PermL *permission-less.* 32, 33

PKI *public key infrastructure.* 24

PoA *proof-of-activity.* 13

PoB *proof-of-burn.* 14

PoS *proof-of-stake.* 13, 14

PoW *proof-of-work.* 11–14, 30, 32

QoS *Quality of Service.* 34, 35, 59

REST *Representational State Transfer*. 34, 49

RPC *Remote Procedure Call*. 60–62, 68

SLR *systematic literature review*. 7, 8, 23, 29

SSD *Solid State Drive*. 74

STXO *spent transaction output*. 10

tps *transactions per second*. 15, 32, 33

UDP *User Datagram Protocol*. 75

URI *uniform resource identifier*. 34

UTXO *unspent transaction output*. 10

Bibliography

[1] K. Panetta, "Top Trends in the Gartner Hype Cycle for Emerging Technologies." Gartner, 2017. `http://www.gartner.com/smarterwithgartner/top-trends-in-the-gartner-hype-cycle-for-emerging-technologies-2017/` [Accessed: 2017-09-29].

[2] A. Dorri, S. S. Kanhere, and R. Jurdak, "Towards an optimized blockchain for iot," in *Second International Conference on Internet-of-Things Design and Implementation*, IoTDI '17, pp. 173–178, IEEE, 2017.

[3] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the internet of things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[4] European Parliament and Council, "Regulation (eu) no 525/2013." Official Journal of the European Union, May 2013. [Accessed: 2017-09-29].

[5] A. Dorri, S. S. Kanhere, and R. Jurdak, "Blockchain in internet of things: Challenges and solutions," *ARXIV.ORG CoRR*, vol. abs/1608.05187, 2016.

[6] D. Wörner and T. von Bomhard, "When your sensor earns money: exchanging data for cash with bitcoin," in *2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, pp. 295–298, ACM, 2014.

[7] M. Conoscenti, A. Vetrò, and J. C. De Martin, "Blockchain for the internet of things: a systematic literature review," in *IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, pp. 1–6, IEEE, 2016.

[8] J. Yli-Huumo, D. Ko, S. Choi, S. Park, and K. Smolander, "Where is current research on blockchain technology?—a systematic review," *PLOS ONE*, vol. 11, no. 10, pp. 1–27, 2016.

[9] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for internet of things: a survey," *IEEE Internet of Things Journal*, vol. 3, no. 1, pp. 70–95, 2016.

[10] B. Kitchenham, "Guidelines for performing Systematic Literature Reviews in Software Engineering." Elsevier, 2007. `https://www.elsevier.com/__data/`

promis_misc/525444systematicreviewsguide.pdf [Accessed: 2017-10-09].

[11] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system." Bitcoin Project, Dec 2008. https://bitcoin.org/bitcoin.pdf [Accessed: 2017-09-15].

[12] F. Tschorsch and B. Scheuermann, "Bitcoin and beyond: A technical survey on decentralized digital currencies," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 2084–2123, 2016.

[13] G. Zyskind, O. Nathan, and A. S. Pentland, "Decentralizing privacy: Using blockchain to protect personal data," in *2015 IEEE Security and Privacy Workshops (SPW)*, pp. 180–184, IEEE, 2015.

[14] Facebook, "Company Info | Facebook Newsroom." Facebook, 2017. https://newsroom.fb.com/company-info/ [Accessed: 2017-12-11].

[15] X. Xu, I. Weber, M. Staples, L. Zhu, J. Bosch, L. Bass, C. Pautasso, and P. Rimba, "A taxonomy of blockchain-based systems for architecture design," in *2017 IEEE International Conference on Software Architecture (ICSA)*, pp. 243–252, IEEE, 2017.

[16] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *2017 IEEE International Congress on Big Data (BigData Congress)*, pp. 557–564, IEEE, 2017.

[17] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *Financial Cryptography and Data Security*, pp. 436–454, Springer, 2014.

[18] R. C. Merkle, *A Digital Signature Based on a Conventional Encryption Function*, pp. 369–378. Springer, 1988.

[19] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger." Ethereum Foundation, 2017. Yellow Paper.

[20] M. E. Peck, "Blockchains: How they work and why they'll change the world," *IEEE Spectrum*, vol. 54, no. 10, pp. 26–35, 2017.

[21] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in *2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 270–282, ACM, 2016.

[22] K. J. O'Dwyer and D. Malone, "Bitcoin mining and its energy footprint," in *25th IET Irish Signals Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014)*, pp. 280–285, IET, 2014.

[23] M. Mueller, “How much power is 1 gigawatt.” Office of Energy Efficiency & Renewable Energy, 2016. https://energy.gov/eere/articles/how-much-power-1-gigawatt [Accessed: 2018-02-25].

[24] Digiconomist, “Bitcoin energy consumption index.” digiconomist.net, 2018. https://digiconomist.net/bitcoin-energy-consumption [Accessed: 2018-06-01].

[25] S. King and S. Nadal, “Ppcoin: Peer-to-peer crypto-currency with proof-of-stake.” White Paper - Peercoin, 2012. http://peerco.in/assets/paper/peercoin-paper.pdf [Accessed: 2018-02-21].

[26] I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld, “Proof of activity: Extending bitcoin’s proof of work via proof of stake [extended abstract]y,” *SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 3, pp. 34–37, 2014.

[27] G. Hardin, “The tragedy of the commons,” *Science*, vol. 162, no. 3859, pp. 1243–1248, 1968.

[28] “Slimcoin: A peer-to-peer crypto-currency with proof-of-burn.” White Paper - Slimcoin, 2014. https://github.com/slimcoin-project/slimcoin-project.github.io/raw/master/whitepaperSLM.pdf [Accessed: 2018-02-21].

[29] M. Swan, *Blockchain: Blueprint for a new economy*, ch. 6. Sebastopol: O’Reilly Media, 2015.

[30] J. Poon and T. Dryja, “The bitcoin lightning network: Scalable off-chain instant payments.” xva-blockchain.com, 2016. https://www.xva-blockchain.com/wp-content/uploads/2017/08/lightning-network-paper.pdf [Accessed: 2018-02-21].

[31] E. Foundation, “Sharding faqs.” GitHub, 2018. https://github.com/ethereum/wiki/wiki/Sharding-FAQs [Accessed: 2018-07-09].

[32] J. Ray, “Mining.” ethereum/wiki Wiki, 2018. https://github.com/ethereum/wiki/wiki/Mining [Accessed: 2018-02-28].

[33] blockchain.info, “Blockchain size.” blockchain.info, 2018. https://blockchain.info/de/charts/blocks-size [Accessed: 2018-02-28].

[34] etherscan.io, “Ethereum chaindata size (geth w/fast sync),” 2018. https://etherscan.io/chart2/chaindatasizefast [Accessed: 2018-02-28].

[35] L. Wang and Y. Liu, “Exploring miner evolution in bitcoin network,” in *Passive and Active Measurement*, pp. 290–302, Springer, 2015.

[36] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of things: A survey on enabling technologies, protocols, and applications,” *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[37] L. D. Xu, W. He, and S. Li, “Internet of things in industries: A survey,” *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[38] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of things (iot): A vision, architectural elements, and future directions,” *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[39] A. Whitmore, A. Agarwal, and L. Da Xu, “The internet of things—a survey of topics and trends,” *Information Systems Frontiers*, vol. 17, no. 2, pp. 261–274, 2015.

[40] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng, “Iot middleware: A survey on issues and enabling technologies,” *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 1–20, 2017.

[41] G. Zyskind, O. Nathan, and A. Pentland, “Enigma: Decentralized computation platform with guaranteed privacy,” *ARXIV.ORG CoRR*, vol. abs/1506.03471, 2015.

[42] Y. Zhang and J. Wen, “An iot electric business model based on the protocol of bitcoin,” in *Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on*, pp. 184–191, IEEE, 2015.

[43] L. Axon, “Privacy-awareness in blockchain-based pki.” Oxford University Research Archive, 2015. Technical Paper.

[44] C. Fromknecht, D. Velicanu, and S. Yakoubov, “Certcoin: A namecoin based decentralized authentication system 6.857 class project.” courses.csail.mit.edu, 2014. `https://courses.csail.mit.edu/6.857/2014/files/19-fromknecht-velicann-yakoubov-certcoin.pdf` [Accessed: 2017-09-22].

[45] S. Huh, S. Cho, and S. Kim, “Managing iot devices using blockchain platform,” in *Advanced Communication Technology (ICACT), 2017 19th International Conference on*, pp. 464–467, IEEE, 2017.

[46] F. Lange, “Release cry uncle (v1.5.2).” GitHub, 2016. `https://github.com/ethereum/go-ethereum/releases/tag/v1.5.2` [Accessed: 2018-05-17].

[47] B. Liu, X. L. Yu, S. Chen, X. Xu, and L. Zhu, “Blockchain based data integrity service framework for iot data,” in *Web Services (ICWS), 2017 IEEE International Conference on*, pp. 468–475, IEEE, 2017.

[48] A. Nordrum, “Mobile world congress 2017: Startup ubirch sails the blockchain into a new application—iot.” IEEE Spectrum, 2017. `https://spectrum.ieee.org/`

tech-talk/telecom/security/mobile-world-congress-2017-startup-ubirch-logs-iot-sensor-data-on-the-blockchain [Accessed: 2018-05-18].

[49] Y. Zhang and J. Wen, "The iot electric business model: Using blockchain technology for the internet of things," *Peer-to-Peer Networking and Applications*, vol. 10, no. 4, pp. 983–994, 2017.

[50] S. Huckle, R. Bhattacharya, M. White, and N. Beloff, "Internet of things, blockchain and shared economy applications," *Procedia Computer Science*, vol. 98, no. Supplement C, pp. 461–466, 2016.

[51] slock.it GmbH, "The usn." slock.it, 2018. https://slock.it/ [Accessed: 2018-05-16].

[52] M. E. Peck and D. Wagman, "Energy trading for fun and profit buy your neighbor's rooftop solar power or sell your own-it'll all be on a blockchain," *IEEE Spectrum*, vol. 54, no. 10, pp. 56–61, 2017.

[53] E. Project, "Enigma's ambition - our latest roadmap." blog.enigma.co, 2018. https://blog.enigma.co/enigmas-ambition-our-latest-roadmap-8d50107ad314 [Accessed: 2018-05-18].

[54] "Cryptocurrency market capitalizations." CoinMarketCap, 2018. https://coinmarketcap.com/[Accessed: 2018-02-21].

[55] A. Nordrum, "Wall street occupies the blockchain - financial firms plan to move trillions in assets to blockchains in 2018," *IEEE Spectrum*, vol. 54, no. 10, pp. 40–45, 2017.

[56] G. Brant, "A next-generation smart contract and decentralized application platform." White Paper - ethereum/wiki Wiki, 2018. https://github.com/ethereum/wiki/wiki/White-Paper [Accessed: 2018-02-21].

[57] J. Ray, "Dagger hashimoto." ethereum/wiki Wiki, 2018. https://github.com/ethereum/wiki/wiki/Dagger-Hashimoto [Accessed: 2018-05-14].

[58] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolic, S. W. Cocco, and J. Yellick, "Hyperledger fabric: A distributed operating system for permissioned blockchains," *CoRR*, vol. abs/1801.10228, 2018.

[59] S. Popov, "The tangle." iota.org, 2017. White Paper.

[60] D. Schiener, "A primer on iota (with presentation)." Official IOTA blog, 2017. https://blog.iota.org/a-primer-on-iota-with-presentation-e0a6eb2cc621?gi=cdeef663fc250 [Accessed: 2018-04-06].

[61] A. Gal, "The tangle: an illustrated introduction - part 5: Consensus, confirmation confidence, and the coordinator)." Official IOTA blog, 2018. https://blog.iota.org/the-tangle-an-illustrated-introduction-79f537b0a455 [Accessed: 2018-04-06].

[62] A. Gal, "The tangle: an illustrated introduction - part 3: Cumulative weights and weighted random walks." Official IOTA blog, 2018. https://blog.iota.org/the-tangle-an-illustrated-introduction-f359b8b2ec80 [Accessed: 2018-04-06].

[63] P. Handy, "Introducing masked authenticated messaging." Official IOTA blog, 2017. https://blog.iota.org/introducing-masked-authenticated-messaging-e55c1822d50e [Accessed: 2018-04-06].

[64] bitcoincore.org, "Bitcoin core integration/staging tree." GitHub, 2018. https://github.com/bitcoin/bitcoin [Accessed: 2018-04-20].

[65] etherscan.io, "Ethereum transaction chart," 2018. https://etherscan.io/chart/tx [Accessed: 2018-04-20].

[66] Ethereum Foundation, "Go ethereum." GitHub, 2018. https://github.com/ethereum/go-ethereum [Accessed: 2018-04-20].

[67] hyperledger.org, "Hyperledger fabric." GitHub, 2018. https://github.com/hyperledger/fabric [Accessed: 2018-04-20].

[68] iota.org, "Iota reference implementation." GitHub, 2018. https://github.com/iotaledger/iri [Accessed: 2018-04-20].

[69] T. T. A. Dinh, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, "Untangling blockchain: A data processing view of blockchain systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. PP, no. 99, pp. 1–1, 2018.

[70] A. Talaminos-Barroso, M. A. Estudillo-Valderrama, L. M. Roa, J. Reina-Tosina, and F. Ortega-Ruiz, "A machine-to-machine protocol benchmark for ehealth applications – use case: Respiratory rehabilitation," *Computer Methods and Programs in Biomedicine*, vol. 129, pp. 1 – 11, 2016.

[71] M. B. Yassein, M. Q. Shatnawi, and D. Al-zoubi, "Application layer protocols for the internet of things: A survey," in *2016 International Conference on Engineering MIS (ICEMIS)*, pp. 1–4, IEEE, 2016.

[72] G. Pardo-Castellote, "Omg data-distribution service: architectural overview," in *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, pp. 200–206, IEEE, 2003.

[73] D. Thangavel, X. Ma, A. Valera, H. X. Tan, and C. K. Y. Tan, “Performance evaluation of mqtt and coap via a common middleware,” in *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pp. 1–6, IEEE, 2014.

[74] J. Benet, “IPFS - content addressed, versioned, P2P file system,” *CoRR*, vol. abs/1407.3561, 2014.

[75] M. S. Ali, K. Dolui, and F. Antonelli, “Iot data privacy via blockchains and ipfs,” in *Proceedings of the Seventh International Conference on the Internet of Things*, IoT ’17, pp. 14:1–14:7, ACM, 2017.

[76] G. S. Ramachandran and B. Krishnamachari, “Blockchain for the iot: Opportunities and challenges,” *ARXIV.ORG CoRR*, vol. abs/1805.02818v1, 2018.

[77] N. Kshetri, “Can blockchain strengthen the internet of things?,” *IT Professional*, vol. 19, no. 4, pp. 68–72, 2017.

[78] J. Ray, “Light client protocol.” GitHub, 2018. https://github.com/ethereum/wiki/wiki/Light-client-protocol [Accessed: 2018-06-05].

[79] K. E. Wiegers, *Software Requirements*. Microsoft Press, 2003.

[80] S. Tilak, N. B. Abu-Ghazaleh, and W. Heinzelman, “A taxonomy of wireless micro-sensor network models,” *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 6, no. 2, pp. 28–36, 2002.

[81] N. Diakopoulos and S. Cass, “Interactive: The top programming languages 2017.” IEEE Spectrum, 2017. https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2017 [Accessed: 2018-06-08].

[82] I. Weber, V. Gramoli, A. Ponomarev, M. Staples, R. Holz, A. B. Tran, and P. Rimba, “On availability for blockchain-based systems,” in *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*, pp. 64–73, IEEE, 2017.

[83] V. Pimentel and B. G. Nickerson, “Communicating and displaying real-time data with websocket,” *IEEE Internet Computing*, vol. 16, no. 4, pp. 45–53, 2012.

[84] K. Hartke, “Observing resources in the constrained application protocol (coap).” Internet Engineering Task Force (IETF), 2015. https://tools.ietf.org/html/rfc7641 [Accessed: 2018-06-19].

[85] E. Bainomugisha, A. L. Carreton, T. v. Cutsem, S. Mostinckx, and W. d. Meuter, “A survey on reactive programming,” *ACM Comput. Surv.*, vol. 45, no. 4, pp. 52:1–52:34, 2013.

[86] K. Hartke and C. Bormann, "The constrained application protocol (coap)." Internet Engineering Task Force (IETF), 2014. https://tools.ietf.org/html/rfc7252 [Accessed: 2018-06-20].

[87] R. Light, "mosquitto.conf man page." mosquitto.org, 2018. https://mosquitto.org/man/mosquitto-conf-5.html [Accessed: 2018-07-29].

[88] D. Gupta and S. Batra, "A short survey on bloom filter and its variants," in *2017 International Conference on Computing, Communication and Automation (ICCCA)*, pp. 1086–1092, IEEE, 2017.

[89] F. Zsolt, "Light ethereum subprotocol (les)." GitHub, 2017. https://github.com/zsfelfoldi/go-ethereum/wiki/Light-Ethereum-Subprotocol-%28LES%29 [Accessed: 2018-06-22].

[90] Ethereum Foundation, "Command line options." GitHub, 2018. https://github.com/ethereum/go-ethereum/wiki/Command-Line-Options [Accessed: 2018-06-22].

[91] C. Svensson, "Filters and events." web3j.readthedocs.io, 2018. https://web3j.readthedocs.io/en/latest/index.html [Accessed: 2018-06-13].

[92] A. Baranov, "Json rpc." GitHub, 2018. https://github.com/ethereum/wiki/wiki/JSON-RPC [Accessed: 2018-06-13].

[93] E. Foundation, "Ether: The crypto-fuel for the ethereum network." ehtereum.org, 2018. https://ethereum.org/ether [Accessed: 2018-06-22].

[94] Ethereum Foundation, "Expressions and control structures." solidity.readthedocs.io, 2018. https://solidity.readthedocs.io/en/v0.4.24/control-structures.html?highlight=require [Accessed: 2018-06-22].

[95] Ethereum Foundation, "Events." solidity.readthedocs.io, 2018. https://solidity.readthedocs.io/en/v0.4.24/contracts.html#events [Accessed: 2018-06-22].

[96] E. Foundation, "Mappings." solidity.readthedocs.io, 2018. http://solidity.readthedocs.io/en/v0.4.24/types.html#mappings [Accessed: 2018-06-22].

[97] M. Saito, "The go-ipfs config file." GitHub, 2018. https://github.com/ipfs/go-ipfs/blob/master/docs/config.md [Accessed: 2018-06-23].

[98] filecoin.io, "Filecoin: A decentralized storage network." filecoin.io, 2018. https://filecoin.io/filecoin.pdf [Accessed: 2018-06-23].

[99] C. Svensson, "Filters and events." web3j.readthedocs.io, 2018. https://web3j.readthedocs.io/en/latest/filters.html [Accessed: 2018-06-13].

[100] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experminetal Design, Measurement, Simulation, and Modelling*, ch. 2. John Wiley & Sons, 1991.

[101] raspberrypi.org, “Raspberry pi 3 is out now! specs, benchmarks & more.” raspberrypi.org, 2016. `https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/` [Accessed: 2018-07-03].

[102] Hardkernel co., Ltd, “Odroid-xu4.” hardkernel.com, 2018. `https://www.hardkernel.com/main/products/prdt_info.php?g_code=G143452239825&tab_idx=1` [Accessed: 2018-07-03].

[103] Intel Corporation, “Intel galileo gen 2 board.” ark.intel.com, 2018. `https://ark.intel.com/products/83137/Intel-Galileo-Gen-2-Board` [Accessed: 2018-07-03].

[104] D. L. Mills, “Internet time synchronization: the network time protocol,” *IEEE Transactions on Communications*, vol. 39, no. 10, pp. 1482–1493, 1991.

[105] J. Han and D. K. Jeong, “A practical implementation of ieee 1588-2008 transparent clock for distributed measurement and control systems,” *IEEE Transactions on Instrumentation and Measurement*, vol. 59, no. 2, pp. 433–439, 2010.

[106] European Comission, “Emissions monitoring & reporting.” ec.europa.eu, 2018. `https://ec.europa.eu/clima/policies/strategies/progress/monitoring_en` [Accessed: 2018-07-22].

[107] Umweltbundesamt, “Internationale berichtspflichten.” umweltbundesamt.at, 2018. `http://www.umweltbundesamt.at/umweltsituation/luft/emissionsinventur/berichtspflichten-international/` [Accessed: 2018-07-22].

[108] A. Peleg, S. Wilkie, and U. Weiser, “Intel mmx for multimedia pcs,” *Commun. ACM*, vol. 40, no. 1, pp. 24–38, 1997.

[109] R. McGill, J. W. Tukey, and W. A. Larsen, “Variations of box plots,” *The American Statistician*, vol. 32, no. 1, pp. 12–16, 1978.

[110] I. Gijbels and I. Prosdocimi, “Loess,” *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 5, pp. 590–599, 2010.