FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Smart Contracts for a Decentralized IoT Data Marketplace

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Michael Sober, BSc
Matrikelnummer 01326403

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Associate Prof. Dr.-Ing. Stefan Schulte

Wien, 1. Juni 2020

Michael Sober	Stefan Schulte

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Smart Contracts for a Decentralized IoT Data Marketplace

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Michael Sober, BSc
Registration Number 01326403

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dr.-Ing. Stefan Schulte

Vienna, 1st June, 2020

Michael Sober	Stefan Schulte

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

Erklärung zur Verfassung der Arbeit

Michael Sober, BSc

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, 1. Juni 2020

Michael Sober

Danksagung

An dieser Stelle möchte ich meinem Betreuer Stefan Schulte danken, der mich bei der Erstellung dieser Arbeit mit seinem sehr hilfreichen Feedback und durch zahlreiche Diskussionen unterstützt hat. Ich möchte auch meinem Berater Salil Kanhere danken, der mich während meines gesamten Aufenthalts an der University of New South Wales betreut und unterstützt hat.

Ich möchte auch meinen Freunden und meiner Familie für die endlosen Dinge danken, die sie getan haben, um mich motiviert zu halten. Ein besonderer Dank geht an meine Freundin, die mich die ganze Zeit unterstützt hat, obwohl ich sie allein gelassen habe und auf die andere Seite der Welt gereist bin, um diese Arbeit zu verfassen.

Acknowledgements

At this point I would like to thank my supervisor Stefan Schulte, who has supported me in the creation of this work with his very helpful feedback and through numerous discussions. I would also like to thank my advisor Salil Kanhere, who provided me with advice and assistance throughout my stay at the University of New South Wales.

I would also like to thank my friends and family for the endless things they did to keep me motivated. Special thanks go to my girlfriend who supported me all the time, even though I left her alone and travelled to the other side of the world to work on this thesis.

Kurzfassung

In den letzten Jahren sind das Internet of Things (IoT) und Blockchain-Technologien immer beliebter geworden. Blockchain-Technologien bieten die Möglichkeit, Transaktionen in einem Logbuch zu speichern, in welchem Daten nur angehängt werden können und das nur schwer manipuliert werden kann. Dieses Logbuch wird von einem Peer-to-Peer-Netzwerk verwaltet. Blockchains der zweiten Generation bieten darüber hinaus die Ausführung von Smart Contracts an. Hierbei handelt es sich um Codeteile, die in der Blockchain gespeichert und von jedem Teilnehmer des Netzwerks ausgeführt werden können. Das IoT wird durch miteinander verbundene Objekte gebildet, wobei ein Objekt jedes Rechengerät sein kann, welches eindeutig adressierbar ist und über standardisierte Protokolle kommunizieren kann. Das IoT wächst stetig, ebenso wie die Menge an Daten, die über das Netzwerk generiert und ausgetauscht werden.

Da die Anzahl der vom IoT generierten Daten weiter zunimmt, gestaltet sich das Auffinden von Datenquellen ohne Datenmarktplatz als sehr schwierig. Zu diesem Zweck bieten Datenmarktplätze eine Plattform auf der verschiedene Parteien ihre Daten anbieten können. Die Kombination von Blockchain-Technologien mit dem IoT bietet vielversprechende Anwendungsfälle, einschließlich dezentraler Datenmarktplätze. Die Forschung hat bereits verschiedene Konzepte und Lösungen im Zusammenhang mit Datenhandel und Datenmarktplätzen hervorgebracht, das heißt sowohl traditionelle Ansätze als auch Ansätze, die bereits Blockchain-Technologien verwenden. Viele dieser Arbeiten decken jedoch nicht alle wesentlichen Funktionen von Datenmarktplätzen ab.

Im Rahmen dieser Arbeit entwerfen und implementieren wir ein Framework für einen dezentralen IoT-Datenmarktplatz. Das Design des Frameworks basiert auf einer Drei-Schichten-Architektur, bei der Smart Contracts verwendet werden, um verschiedene Funktionen zu implementieren und die Regeln des Datenmarktplatzes durchzusetzen. Zu diesem Zweck wurden unter anderem mehrere Smart Contract-Plattformen miteinander verglichen, um festzustellen, welche Unterschiede bestehen und welche für diese Anwendung am besten geeignet ist. Darüber hinaus enthält das Framework grafische Benutzeroberflächen, einen Proxy, der es Anbietern und Verbrauchern ermöglicht, IoT-Geräte zu integrieren und einen Broker, der den Datenhandelsprozess erleichtert und ressourcenintensive Aufgaben übernimmt. Abschließend evaluieren wir die Kosten, die durch die Verwendung von Smart Contracts entstehen und diskutieren Probleme, die während der Implementierung aufgetreten sind.

Abstract

In recent years, the IoT and blockchain technology have become increasingly popular. Blockchain technology offers the possibility to store transactions in an append-only and tamper-evident log, which is managed by a peer-to-peer network. Second-generation blockchains also offer the execution of smart contracts, which are pieces of code that can be stored on the blockchain and executed by every participant of the network. The IoT is formed by interconnected objects, whereby an object can be any computational device that is uniquely addressable and is able to communicate over standardized protocols. The IoT is growing steadily and so is the number of data that is generated and exchanged over the network.

As the number of data generated by the IoT continues to increase, finding data sources without a data marketplace becomes very difficult. For this purpose, data marketplaces provide the platform to enable different parties to trade their data. The combination of blockchain technology with the IoT offers promising use cases, which include decentralized data marketplaces. Research has already brought up several concepts and solutions related to data trading and data marketplaces, i.e., both traditional approaches and approaches that already utilize blockchain technology. However, many of these works do not cover all the essential functionalities of data marketplaces.

In the course of this work, we design and implement a framework for a decentralized IoT data marketplace. The design of the framework is based on a three-tier architecture, whereby smart contracts are used to implement various functionalities and enforce the rules of the data marketplace. For this purpose, among other things, several smart contract platforms were compared with each other in order to determine their differences and which one is best suited for this application. Furthermore, the framework includes Graphical User Interfaces (GUIs), a proxy which enables providers and consumers to integrate IoT devices and a broker which facilitates the data trading process and takes over resource intensive tasks. Finally, we evaluate the costs that arise from the use of smart contracts and discuss problems that occurred during the implementation.

Contents

CHAPTER 1

Introduction

1.1 Motivation and Aim of the Work

The IoT describes a world-wide network of interconnected objects, which are uniquely addressable and communicate over standardized protocols [AIM10]. As the number of IoT objects rise rapidly, there is also a rapid growth of data that is generated and transmitted over the network by those objects. In many cases, the amount of data that is generated by an organization rises faster than the amount of data that can be processed [ZPG12].

Data marketplaces allow organizations to exchange (large amounts of) data for free, as it is the case in open (government) data scenarios [JCZ12] or based on a cost model [CPV+16]. In many scenarios, data is very valuable and thus interest in making economic benefit from it is very large. Studies have shown that the potential annual value is in the range of hundreds of billions US dollars in certain application areas [ZPG12].

Centralized approaches of creating a data marketplace come with different challenges concerning scalability issues, expensive infrastructure or single points of failure [GKJ18]. With regard to IoT data, one particular issue with centralized approaches is that IoT devices lack the resources to maintain multiple connections to other parties, e.g., because of energy constraints [MŽ16]. One possible way to cope with these problems would be to explore solutions which follow a decentralized approach. Within this thesis, it is the goal to explore if blockchain technology could be adapted in order to realize such a decentralized marketplace for the IoT.

In recent years, blockchain technology gained popularity especially through its usage as the technical backbone for cryptocurrencies like Bitcoin [N+08]. A blockchain is a linked list of blocks, where the links are hash pointers [NBF+16]. Blockchains provide properties such as public verifiability, transparency, integrity, and redundancy [WG18]. Second-generation blockchains offer not only the possibility to exchange tokens, but also to execute smart contracts. Such contracts are scripts which get stored on the blockchain.

A smart contract can be invoked by sending a transaction to it, which results in the automatic and independent execution of the script by every node in the network [CD16]. For a decentralized IoT data marketplace, smart contracts could provide the means to enable trustless data trading. A blockchain could then store the history of all trades that occurred in order to make data exchanges verifiable [GKJ18]. Consequently, we will take a closer look at this possibility in the course of this work.

The expected result of this thesis is the conceptualization and implementation of a marketplace for trading IoT data [GKJ18]. The marketplace will be realized as a software framework which follows a decentralized approach where no trusted third party needs to be involved. For this, we will investigate if the utilization of peer-to-peer (P2P) and blockchain technology is providing the necessary technological functionalities to realize a data marketplace as described above. The design of the framework will build upon a concept proposed by Gupta et al. [GKJ18], and will extend the existing conceptual model by more concrete technical aspects. The framework will provide a solution to utilize the ever-growing amount of data generated by IoT devices. Data producers will get the possibility to obtain a higher economic benefit by making the data accessible for arbitrary data consumers. The data consumers will be able to buy data to use it for their own purposes (e.g., to enhance products and services). The following paragraphs give a brief overview on the goals of this thesis.

Smart Contracts and Blockchain Technology The framework should use smart contracts and blockchain technology to implement data trading. Thus, an important goal of this work is to determine if these technologies are suitable for the intended goals. The thesis should also give further insight into blockchain technology and different smart contract platforms. Possible examples of such platforms investigated during the course of the thesis are Ethereum[1], Neo [2] and Hyperledger[3]. As these are just a few examples, an overview of different platforms will be created. The overview will also be useful to select the smart contract platform to be used for the implementation of the framework.

Decentralized IoT Data Marketplace The main focus of this work is the design and implementation of the decentralized IoT data marketplace, which provides a framework for data exchange within the IoT. The framework should enable the implementation of decentralized IoT data trading based on smart contracts. Different challenges of exchanging very large amounts of data will have to be considered for the design and implementation of the framework.

[1] https://www.ethereum.org/
[2] https://neo.org/
[3] https://www.hyperledger.org/

Evaluation of the Framework Another important goal of this work is the evaluation of the framework. Hereby, the examination of the costs which arise from the use of the implemented smart contracts is particularly important in order to be able to estimate when it is worth using them. Furthermore, possible limitations and weaknesses of smart contracts which are observed during the implementation should be discussed.

1.2 Methodology and Approach

The methodological approach pursued in this work can be divided into four parts. Each of these parts is briefly explained in the following paragraphs.

Literature Work The first step of this thesis is to gather background information about the underlying technologies and the related work in the area of data marketplaces. Various issues will play a major role when deciding how the framework will be designed and implemented. Amongst these issues are the consideration of problems with existing solutions and the selection of the smart contract platform, just to give a few examples.

Design The design of the framework will build upon the work towards a framework for a decentralized IoT data marketplace proposed by Gupta et al. [GKJ18]. Based on the information gathered in the literature work, the design will be extended and refined.

Implementation Taking the results of the literate work and the design into consideration, a suitable technology stack will be selected to implement the framework. Hereby, the selection of the right technology stack plays an important role to make sure that the implementation meets predefined functional and non-functional requirements.

Evaluation In the final part of the thesis, the implementation needs to be tested and evaluated. We are going to look at the costs which arise from the use of the implemented smart contracts and discuss how smart contracts are suitable for implementing a decentralized IoT data marketplace.

1.3 Structure

This thesis is structured as follows

Chapter 2 introduces the background knowledge that is needed for the design and implementation of a decentralized IoT data marketplace.

Chapter 3 discusses related work in the area of data marketplaces, including centralized and decentralized solutions, machine-to-machine-payments and application areas of blockchain technology in the IoT.

Chapter 4 explains different smart contract and decentralized application platforms and their differences. The differences are shown by looking at nine different characteristics.

Chapter 5 shows the requirements development process of the framework for a decentralized IoT data marketplace including user classes, use cases, system features and quality attributes.

Chapter 6 covers the design of the framework for a decentralized IoT data marketplace. In the course of this, design decisions are explained and substantiated, and the exact architecture of the system is described.

Chapter 7 describes how the previously designed framework is implemented and explains implementation specific details, including the used technologies.

Chapter 8 contains the evaluation of the framework. The evaluation shows which costs arise from the deployment and usage of the implemented smart contracts and discusses limitations and possible weaknesses of smart contracts which were observed during implementation.

Chapter 9 represents the final chapter of this work. It concludes the results of this thesis and follows up with possible future work in this research area.

CHAPTER 2

Background

Some background knowledge is needed for the design and implementation of a decentralized IoT data marketplace. Since we want to commit ourselves explicitly to a data marketplace for the IoT, we first have to take a closer look at the IoT (see Section 2.1). Here, we will discuss how the term first came up and then we will take a closer look at several essential building blocks of the IoT. Furthermore, we will deal with the technologies behind it, which make the IoT possible in the first place. Also we will define what a data marketplace is and what benefits they offer (see Section 2.5).

In addition, we want to state the rationale behind choosing a decentralized architecture to realize a data marketplace. We want to realize data trading by using blockchain technology, thus we are first going to introduce the basics of cryptocurrencies and blockchain technology (see Section 2.2). In the course of this, we will take a closer look at cryptographic basics needed to realize a cryptocurrency including digital signatures and hash pointers. Cryptocurrencies are an important part of many smart contract and decentralized application platforms and therefore need to be discussed beforehand. The basic structure and the concept of blockchain as a data structure is considered in more detail, followed by blockchain as an append-only distributed ledger of transactions managed by a peer to peer network. Blockchains are used in all subsequently introduced smart contract and decentralized application platforms (see Chapter 4) as a tamper-evident log of transactions.

After that, we will take a look at how the idea of smart contracts came to live, and what smart contracts are all about (see Section 2.3). Since the idea of smart contracts came up long before blockchain technology became an important topic, we will explain smart contracts in the area of blockchain technology in addition to the basic idea (see Section 2.4). Finally, we will look at the potential of blockchain technology in the area of the IoT to identify potential applications and challenges.

2.1 The Internet of Things

The IoT describes a world-wide network of interconnected objects, which are uniquely addressable and communicate over standardized protocols. In short, the term IoT refers to a combination of different visions and technologies, which all together form this paradigm [AIM10]. The term IoT was most likely used first by Kevin Ashton in 1999 [A+09]. He mentioned that one of the biggest problems of the internet is that it relies mostly on humans to get information. The problem with this concept is that humans only have limited resources. If computers were able to communicate and cooperate autonomously with other things, instead of relying on human input, it would greatly improve our perceptions of the environment, and thus the interaction as well.

The authors in [AFGM+15] define several essential building blocks for the IoT: Identification, Sensing, Communication, Computation, Services and Semantics. Each of these building blocks contains important aspects which are necessary for the realization of the IoT and are backed by different enabling technologies.

A crucial part to realize the IoT includes the identification of objects. In [JFFL12], the authors mention Radio-frequency Identification (RFID) technology as a fundamental part of the IoT. It offers the possibility to uniquely identify objects that are equipped with radio tags. RFID readers query the tags by sending a query signal, which triggers the transmission of the identifier from the surrounding tags. A RFID tag is basically a microchip, that consists of an antenna, integrated circuit and a circuit board. There are many different forms of RFID tags which are categorized in different classes. Most RFID tags are passive, which means they will use the energy provided from the query signal to transmit the response. The query signal is used to generate the current through induction to power the RFID tag. The opposite is the case with active RFID tags, which have a built-in power source to power the microchip. Mixed forms like semi-active RFID tags are also possible. The tags operate on different frequency bands from low frequency 124–135 kHz up to ultra high frequencies (UHF) at 860–960 MHz [AIM10, JFFL12].

Sensing in the IoT means the collection of data from different objects that are capable of perceiving their surroundings in the real world. The collected data can be analyzed and used to decide which action should be taken [AFGM+15]. In this regard, sensor networks are especially worth mentioning. These networks consist of a large number of sensor nodes that are able to communicate with each other in a wireless fashion and have enough computing power to realize self-organizing capabilities. This allows the sensor nodes to be deployed in terrains, which would otherwise be hardly accessible. Since the sensor nodes are usually very densely deployed, wireless multi-hop communication should be used for low power consumption as this is a major concern in wireless sensor networks [ASSC02]. The majority of today's sensor area networks use the IEEE 802.15.4 standard for Wireless Personal Area Networks (WPAN). Wireless sensor networks offer a wide range of applications in transportation and logistics, healthcare, smart environments and many more [AIM10]. In [MCP+02], the authors describe the use of wireless sensor networks for real-world habitat monitoring and showcase the potential of this technology.

The realization of communication in the IoT plays a major role due to the differences to the Internet Protocol (IP)-based internet. Devices are subject to certain restrictions that must be observed. Networks contain a much larger number of participants, which may be out of the range of IP addresses, whereby IPv6 already counteracts this problem. The packet size in the IEEE 802.15.4 standard is also much smaller in size and most of network participants will spend their time in idle mode to reduce the power consumption to a minimum [AIM10]. Typical communication protocols for the IoT include WiFi, Long Term Evolution (LTE)-A, Bluetooth Low Energy (BLE), IEEE 802.15 and the already mentioned RFID. Apart from these, many other communication protocols for the IoT exist [AFGM+15].

Different hardware platforms provide the means for computations in the IoT. These are necessary to realize IoT applications which not only act as data collectors, but are also able to perform intelligent actions on their own, which enables devices to directly collaborate with each other. Apart from the typical hardware platforms like Arduino and Raspberry PI, also software platforms specifically for the IoT provide additional functionality [AFGM+15].

IoT services can be divided into four different categories. Identity-related Services which act as the link between the physical and the virtual world to identify objects. Information Aggregation Services which collect the raw data from devices and are responsible for data processing. Collaborative-Aware Services that use the processed data and in turn analyze it to make certain decisions and take actions based on those decisions and finally Ubiquitous Services, which should fulfill the anytime, anywhere and anyone aspect of the IoT [AFGM+15].

The last essential building block refers to the semantics in the IoT, which describes the importance of knowledge extraction to make the right decisions based on the analyzed data. This encourages the use of semantic web technologies [AFGM+15].

2.2 Cryptocurrencies and Blockchain

The first cryptocurrency Bitcoin was introduced 2008 by Satoshi Nakamoto [N+08]. He envisioned an electronic payment system without the need of trusted third parties and clarified the weaknesses of the traditional trust-based model, where financial institutions process the payments. The use of cryptography, in particular digital signatures allow two parties to directly interact with each other. It provides the means to protect participating stakeholders from fraud and to create irreversible transactions. His solution to the double spending problem, was the introduction of the Proof-of-Work (PoW) algorithm, which will be considered later in more detail. A major aspect is to make tampering with transactions really expensive, to make it practically not worthwhile.

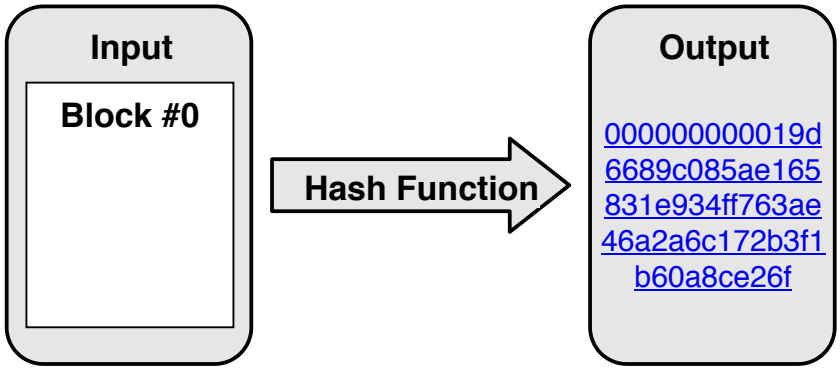

Figure 2.1: A hash function that takes a block as input and produces a string of specific length as output. The example output is the hash value of the genesis block of Bitcoin.

2.2.1 Hash Functions and Digital Signatures

Cryptocurrencies use certain cryptographic primitives to secure the system. One of these essentials is the cryptographic hash function. To define a cryptographic hash function it is essential to define the general hash function first. A hash function (see Figure 2.1) is a mathematical function, which can take a string of any size as input, produces a fixed-size output and is efficiently computable [NBF+16].

In order to turn a general hash function into a cryptographic hash function, three properties have to be ensured. A cryptographic hash function has to be collision-resistant. It must be infeasible to find two inputs that produce the same output. A cryptographic hash function must also fulfill the hiding property. Given two output values of the hash function there must be no practicable way to compute the corresponding input values. If the set of input values is not spread-out, the hiding property can be ensured by concatenating the input with an additional input whose value is from a set that is wide spread-out. The last property, which has to be fulfilled by a cryptographic hash function is puzzle friendliness. It must be very difficult to find another value that concatenated with the input produces a result that is in the same target space. Later, it will be seen that this property plays an important role especially for the PoW algorithm [NBF+16].

Hash functions can be implemented in various ways, some of which have been standardized. The National Institute of Standards and Technology (NIST) standardized different cryptographic hash functions [Dan15], including the Secure Hash Algorithm (SHA)-256, which finds application in various cryptocurrencies (including Bitcoin and Neo), although this is already an older standard. The successor was introduced through the SHA-3 competition, where Keccak was announced as the winner using sponge construction [BDPVA13].

In addition to cryptographic hash functions, cryptocurrencies also make use of digital signatures [NBF+16]. They form the counterpart to traditional signatures and must ensure two important properties. A digital signature must be easy to verify and only the

signatory should be able to create it. To create a digital signature a key pair needs to be generated. One key will be kept secret and is used to create the actual signature, whereby the public key can be used to verify the correctness of the signature. The signature must be existentially unforgeable and valid signatures must be verifiable given the public key and the original message [NBF+16]. Cryptocurrencies like Bitcoin, Ethereum and Neo make use of the Elliptic Curve Digital Signature Algorithm (ECDSA) standard to create digital signatures. The algorithm is based on the fact that no subexponential-time algorithm is known for the elliptic curve discrete logarithm problem [JMV01].

2.2.2 Blockchain

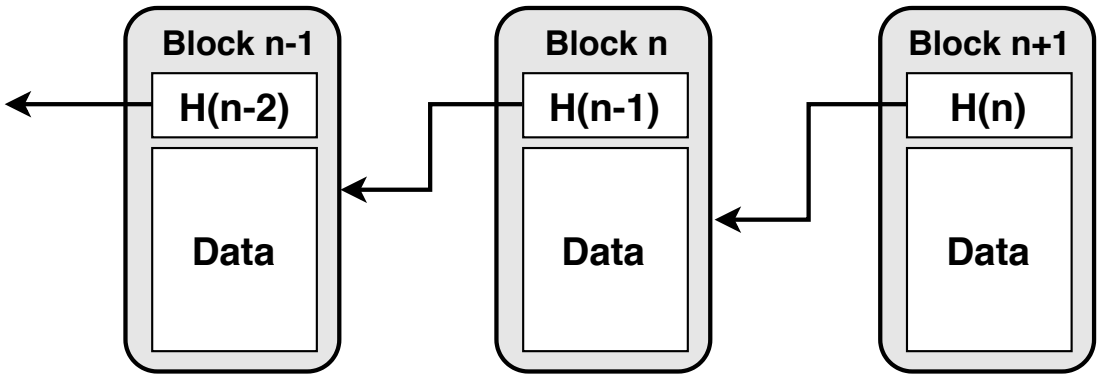

Figure 2.2: A linked list of blocks utilizing hash pointers [NBF+16]

Most cryptocurrencies use a blockchain as their underlying data structure to record transactions in an append-only and tamper-evident log. A blockchain is a linked list that utilizes hash pointers (see Figure 2.2). A hash pointer is not only a pointer to the location of the stored information, but also includes the hash value of the stored information. This offers the advantage that besides the location, we can also verify the data. A variety of data structures can be implemented by using hash pointers, of which we have already mentioned the blockchain [NBF+16].

With the help of hash pointers, we can ensure the tamper-evident property of blockchains. If an adversary changes the information of one element, it must change all hash pointers of the following elements (see Figure 2.3). An additional data structure, which uses hash pointers and is used for different cryptocurrencies, are Merkle trees. A Merkle tree is a binary tree that uses hash pointers, where the data is stored in the leaves of the tree, with each parent containing the hash values of its children. This data structure offers the advantage to efficiently check membership or non-membership of an element by providing just the path of the root to the element [NBF+16].

Other works refer to blockchains not only as a simple data structure that can be stored on a single machine, but rather an append-only distributed ledger of transactions managed by a peer-to-peer network. Decentralization is a major part of blockchain technology. Every node in the system is responsible for verifying transactions and storing them in

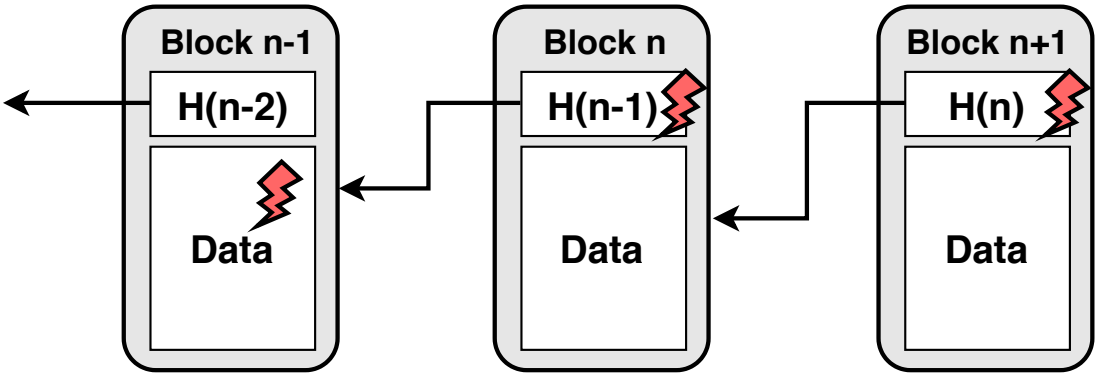

Figure 2.3: Modified data invalidates the hash pointers of the following blocks in the blockchain [NBF+16]

its own copy of the distributed ledger. Blockchain systems can differ in numerous ways from each other. There are a lot of different properties, which can be considered when comparing various blockchains. A major distinction is made, by how the system performs membership selection, to select the participants which are allowed to verify and propose the next block which should be included in the blockchain. One can differentiate between permissionless, permissioned and partially permissioned blockchains [NYGEV19]. In a permissionless system, every node is allowed to join the network. The opposite is the permissioned system, where a node has to get the permission to join the system. There are also hybrid systems, which allow every node to join the network, but only certain nodes are allowed to take part in the consensus mechanism. Meanwhile, there are countless different membership selection mechanisms in use, with the most popular being the PoW mechanism introduced by Satoshi Nakamoto in Bitcoin, which relies heavily on membership selection by computing power. Many other membership selection mechanism emerged like Proof-of-Stake (PoS), Proof-of-Authority (PoA) or Proof-of-Capacity (PoC). Another important distinction can be made by the consensus mechanism. The consensus mechanism is responsible for deciding which block to add next. Besides the consensus mechanisms used in Bitcoin and Ethereum, which will be explained later on, there are plenty other mechanisms in use [NYGEV19].

2.2.3 Authentication and Authorization

Decentralized identity management plays a major role for cryptocurrencies. A user must be able to create identities on his own. In Bitcoin, the addresses act as identities. We have already mentioned that digital signatures need a key pair which consists of the private and the public key. An address is nothing more than the hash of the public key of the user. To create a new identity, a user can simply generate a new key pair. Each transaction has to be signed by a user with its private key to produce a valid signature to ensure that the user is authorized to spend the coins specified in the transaction [NBF+16].

Of course, there are systems that do not value decentralization. There are other issues to consider when it comes to authentication and authorization. This will be seen later when we take a closer look at a permission-based system.

This should only give an idea of the basics of how to build a simple cryptocurrency: The blockchain constructed by linking blocks with the usage of hash pointers will be used as the underlying data structure to build an append-only distributed ledger, which preserves the history of all transactions that occurred. Digital signatures provide the means to verify the transactions. As we will see in Section 4.2, Cryptocurrencies like Ethereum, also rely on further aspects, e.g., the execution of smart contracts.

2.3 Smart Contracts

Smart contracts have already been proposed as early as 1994 by Nick Szabo [Sza94]. He defined smart contracts as „a computerized transaction protocol that executes the terms of a contract“. This offers the possibility to reduce the risk introduced by trusted intermediaries and minimize errors, whether they were made intentionally or accidentally. Already back then, he mentioned the potential of smart contracts for digital cash protocols, to prevent fraud and reduce the costs of commercial transactions. In 1997 he further explained his idea of smart contracts and described possible real-life examples [Sza97]. The embedding of the contracts should make it expensive to break a contract, thus making it unrewarding for the breacher. Smart contracts provide the means for greater security of a system through better observation and verification by self-enforcing predefined rules.

In the area of blockchain technology and cryptocurrencies, smart contracts are pieces of code respectively programs, which get stored on the blockchain and executed by multiple or rather every node of the network. Already Bitcoin scripts can be seen as smart contracts, which are however very limited. In general a smart contract is stored on the blockchain and has a unique address and its own state. The unique address is used to invoke the functions of a smart contract, which either reads or manipulates the state. Every participant of the network is able to view the smart contract, since it is stored directly on the blockchain, although there are exceptions to this, as will be seen later on regarding Hyperledger Fabric (see Section 4.4). This promotes confidence and trust, since the code can be reviewed by everyone, to check for malicious behaviour. Smart contracts are very long-lasting, since they will forever reside on the blockchain. This has to be considered when developing decentralized applications. After the invocation of a smart contract, multiple nodes or rather every node on the network compute the outcome independently in a deterministic manner [CD16].

2.4 Blockchains and Smart Contracts for the IoT

According to the authors of [FCFL18], today's IoT solutions mainly focus on cloud services and have a centralized architecture. At some point in the future this solutions will not be suitable anymore, due to the enormous growth of the IoT. Centralized solutions suffer from high infrastructure and maintenance costs and require trust in participating partners. This could be remedied by blockchain technology, but should not be mistaken as a silver bullet to all problems. Developers of IoT solutions, among other things, have to ask a variety of questions to see if blockchain technology can actually add value to a problem. Numerous IoT solutions based on blockchain technology can be identified in different application areas like smart cities, farming, energy, transportation and many more. Christidis and Devetsikiotis further elaborate how the combination of blockchains and smart contracts with the IoT opens new possibilities for decentralized applications and business models [CD16]. The blockchain brings a whole new opportunity by providing the means for devices to interact with each other in a trustless network.

One interesting use case offers manufacturers to replace their centralized solutions for firmware updates with a decentralized architecture, that is able to cope with the fast growing amount of different devices. Instead of providing firmware updates via a centralized server of the manufacturer, a smart contract could be used to store the hash of the binary on the blockchain. The binary itself is stored in a distributed peer-to-peer filesystem. Devices could automatically interact with the smart contract to retrieve the hash of the binary and download it from the distributed filesystem. New devices would be able to update their firmware even after the manufacturer would not provide updates anymore [CD16].

The combination of blockchain and the IoT provides a lot more interesting use cases. Further examples describe the usage in the area of supply chain management, to transparently track assets by using the blockchain as a shared distributed database, which is also cryptographically verifiable [CD16]. The International Business Machines Corporation (IBM) also offers blockchain-services for tracking high-value items across supply-chains [Ksh17]. Multiple startups use blockchain technology in supply-chain management for trust and transparency. To further improve security in the IoT, blockchains can be used for access and identity management systems. The properties of a blockchain, such as immutability are very beneficial and provide better security against spoofing attacks [Ksh17].

Another use case that is especially important for this work is the usage of blockchain technology to facilitate trading of services and data between devices. Cryptocurrencies enable every device to be a merchant of their own resources. Devices would be able to buy or sell collected data, offer the possibility to rent disk space or even provide access to different services [CD16]. In later chapters of this work, we will focus on trading data and exploring how a data marketplace for the IoT can be implemented, by using blockchains and smart contracts.

Besides the interesting use cases that were discussed earlier and the useful features regarding decentralization, anonymity and security that the blockchain brings to the IoT, we also have to consider some challenges. Blockchain networks that make use of the PoW algorithm require a high amount of computation power, while the resources of IoT devices are usually very restricted. This causes the mining process to be very time consuming, which is often not desirable for low-latency IoT applications. Scalability is another important factor that is a major concern regarding PoW blockchains. This requires new approaches that may take a different attempt. Certain IoT devices may be restricted through their bandwidth, whereby protocols which make use of a blockchain require higher bandwidth. Not only the advantages but also the disadvantages of a blockchain system have to be considered [DKJ16].

2.5 Data Marketplaces

Data is an extremely valuable asset in today's society. From individuals, small organizations to large companies, everyone can benefit from this development. Organizations tend to collect a large amount of data to improve their products and processes in order to gain higher economic value. Google, for example, uses collected user behavior data for ad targeting to improve quality, improve user experience and maximize revenue [CSC+05]. Now if we move away from user behavior data and look at data generated by millions of different IoT devices around the globe, we can see that there is a lot of potential behind it. The number of IoT connected devices is growing rapidly and so is the amount of data generated by these devices. Statistics show that the number of connected IoT devices is expected to rise from 26.66 billion in 2019 to 75.44 billion worldwide by 2025 [IHS16]. Often, so much data is generated that a single organization is not able to process all of it and much potential is lost [ZPG12]. In order to utilize this unexploited potential, data marketplaces are becoming increasingly attractive.

Data marketplaces act as a platform for trading data between one or more entities. Organizations can use data marketplaces to exchange (large amounts of) data for free as it is the case in open (government) data scenarios [JCZ12] or based on a cost model [CPV+16]. Even individuals can benefit from data marketplaces by personally offering their own data for sale or by receiving access to premium services or compensation from organizations to share their data. One application area would be in the field of fitness and health data. Smart watch manufacturers may ask their users if they can share their fitness or health data collected through their smart watches and offer some form of compensation in exchange. Machine-to-machine economy is also becoming increasingly interesting, with various devices automatically providing data and services in exchange for money. Smart cables paying for electricity provided by smart sockets [LDBA17] or vehicles paying for fuel at the gas station [HHL18] give a good example regarding this topic. Data marketplaces would provide additional means to enable machine-to-machine economy by providing a platform for machines to exchange their data with each other for money.

The first efforts in the direction of data marketplaces for the IoT were presented with the Sensing-as-a-Service (SaaS) model. This model emerged from the Everything-as-a-Service (XaaS) model, which for example in the context of the cloud, includes services for computing power, computing platforms and on-demand software [DFZ+15]. SaaS mainly describes the sale of sensor data generated by IoT devices. Owners of sensing devices can sell the generated data in exchange for a fee or get compensation in another form. Meanwhile, there are already many different proposals for the realization of data marketplaces, whether data marketplaces specialised on specific data for example IoT data marketplaces or data marketplaces for general data exchange. We can further distinct between data marketplaces based on a centralized architecture [MŽ16] or a decentralized architecture [GKJ18]. Data marketplaces can be open to anyone to participate or restricted to only certain organizations and individuals. In this work, we will focus on creating a decentralized IoT data marketplace based on blockchain technology. This allows us to trade data without having to trust a third party, because every participant of the network is going to verify all transactions. The rules of the data marketplace would be enforced automatically by all participants. Furthermore, decentralized data marketplaces, which use a public distributed ledger, provide transparency for all transactions, which makes it easier to verify data exchanges in retrospect. However, the scaling problem that many of todays smart contract and decentralized application platforms suffer from, is a major drawback. Thus, much effort is spent in this area to solve it [CMVM18]. In order to facilitate payments in a centralized data marketplace, we would need to trust payment networks, which are responsible for the entire processing of payments. The owner of a centralized data marketplace acts as a central authority, which is in charge of the rules of the system and can change it arbitrarily, only considering his or her own interests. Additionally transparency is also limited to the owner of the data marketplace. To cope with the huge amounts of data, an individual or an organization would need to build an enormous infrastructure.

CHAPTER 3

Related Work

Research on data marketplaces has led to a number of concepts and solutions. In Chapter 2, we have already clarified what a data marketplace is and briefly explained related work. To get a deeper insight into already existing solutions and concepts, we examine them further in the course of this chapter. The related work includes traditional data trading concepts (see Section 3.1), including the SaaS model and various solutions of centralized IoT data marketplaces. Furthermore, we discuss blockchain-based data trading concepts (see Section 3.2), covering blockchain-based payment protocols including thing-to-thing payments and solutions for decentralized IoT data marketplaces. Nevertheless, there is still the need to conduct additional research and development in this area, as discussed in Section 3.3.

3.1 Traditional Data Trading

3.1.1 Sensing as a Service

Zaslavsky et al. propose the SaaS model, whereby sensor data should be provided for a fee. Furthermore, the authors describe the importance of big data and its three main characteristics: Volume, variety and velocity. The analysis of big data allows organizations to improve their services, products and decision making process [ZPG12].

Kaisler et al. discuss three issues when handling very large amounts of data (as is also necessary on an IoT data marketplace): (i) Storage and transport issues regarding the limitations of current disk technology and communication networks. (ii) Management issues due to different protocols, data sources and data formats, and (iii) processing issues due to the quantity of data [KAEM13].

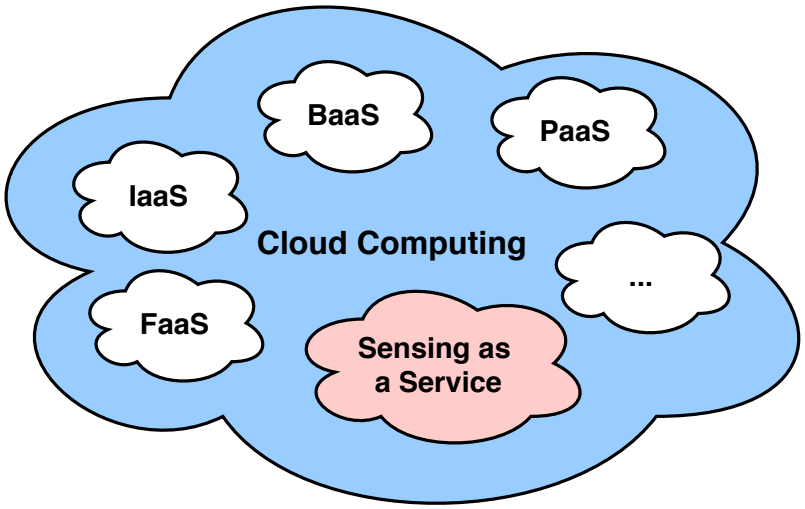

Figure 3.1: Cloud computing including the Sensing as a Service model

A problem that the authors of [ZPG12] also have found, however, is that the amount of data an organization can analyze is declining. This is based on the statement that organizations simply do not have the necessary technology to analyze and process the sheer amount of data. That is where the proposed SaaS model can help to make use of data that otherwise would not be used. As we have previously mentioned, this model is one of the XaaS models which are an important part of cloud computing (see Figure 3.1). The authors identify three main participants within this model: The owners of the sensors, mediators for publishing and managing sensors, and the data consumers. A smart fridge scenario explains how owners could publish their data for food manufacturers which in return provide the sensor owner with a fee or a discount [ZPG12].

Similar to Zaslavsky et al. [ZPG12], Perera et al. [PZCG14] propose a SaaS model, however, for smart cities. The model consists of four conceptual layers. The first layer comprises sensors and sensor owners, whereby sensors are categorized into different categories based on the ownership of the sensors. The second layer consists of the service providers, which manage available sensors and operate as the link between sensor owners and sensor consumers. The extended service providers form the third layer, which provide value-added services to sensor consumers, such as sensor selection based on requirements. The final layer consists of the sensor data consumers which have to register themselves and obtain a digital certificate to interact with the (extended) service providers to consume sensor data [PZCG14]. In addition to the smart home scenario, which is discussed in [ZPG12], the authors further explain possible application areas especially important for smart cities.

For instance, SaaS could be used to facilitate better waste management. Several stakeholders are affected or can be affected by waste management which all would have to deploy their own sensors. The SaaS model would allow these stakeholders to share the data between each other, which results in great cost reductions. Different sensors are deployed in various locations and the data is uploaded to the cloud. The authors further investigate the usage of the SaaS model for smart agriculture. Sensors collect information about experimental crops, whereby the application of SaaS allows different research organizations to access the data. The main advantages of this model include built-in cloud computing, workload distribution, sharing and reusing of data, lower data acquisition cost and the collection of previously unused data [PZCG14].

Nevertheless, some open challenges and issues must be considered, for example, architectural design, standardization, trust, security and privacy [PZCG14]. Among other things, the use of blockchain technology could remedy this situation, as will be further discussed in Section 3.2.

3.1.2 Centralized Data Marketplaces

Schomm et al. [SSV13] conducted an initial survey on data marketplaces in 2013. The authors define a data marketplace as "a platform on which anybody (or at least a great number of potentially registered clients) can upload and maintain data sets". The authors examined data providers based on twelve dimensions: Type, Time Frame, Domain, Data Origin, Pricing Model, Data Access, Data Output, Language, Target Audience, Trustworthiness, Size of Vendor, and Maturity. Each of these dimensions capture different properties of data marketplaces which can be used to differentiate between various data vendors. In the following discussions, we will focus on IoT data marketplaces.

Mišura and Žagar propose a model of an IoT data marketplace [MŽ16]. By creating a data marketplace, the authors identify several advantages that also coincide with the advantages of the SaaS model. They also mention that the design of existing solutions of data marketplaces on the internet like Microsoft's Azure Data Market (already retired [Ram16]) and Salesforce's data.com are not well-suited to implement a marketplace for IoT data. These marketplaces only offer collections of data that were previously acquired. IoT data marketplaces require a special design taking into account the characteristics of data generated by IoT devices. That is why the authors have developed their own model.

The model is based on a centralized architecture (see Figure 3.2). IoT devices and data consumers register themselves at the marketplace through a web interface. Sensor measurements of devices are saved in a database. Devices never communicate directly with consumers to ensure low battery usage. For consumers to retrieve the data, a query mechanism is used: Consumers create a query in which certain information like type, time and budget must be provided. Devices which satisfy the query are returned by the system and can be accepted by the consumer [MŽ16].

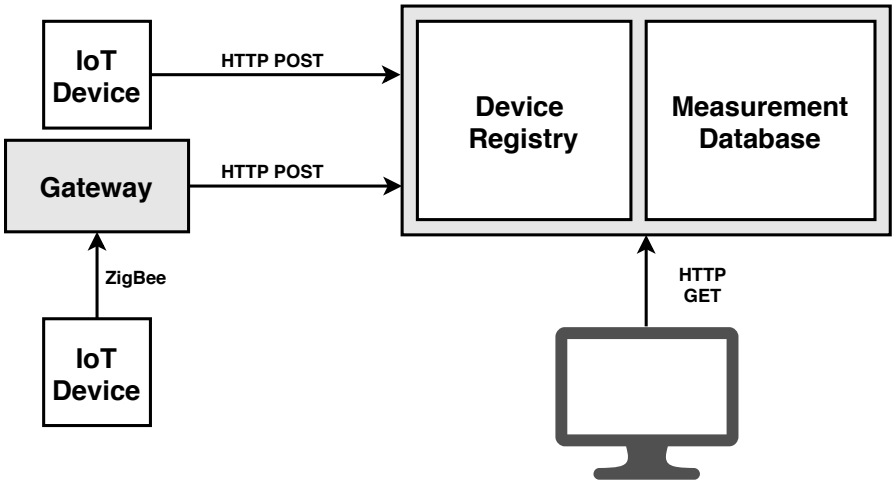

Figure 3.2: System architecture proposed by Mišura and Žagar [MŽ16]

Furthermore, the authors of [MŽ16] consider another important aspect. To ensure the fairness of the system, each device has a unique credibility value which is calculated by a simple formula. The formula takes under considerations how many measurements were completed and how many were agreed to. This value can then be used to filter out devices that do not meet the needed fairness requirements of a customer. While this solution is based on a centralized approach, it gives important insights into design issues for IoT data marketplaces.

In [CPV+16], the authors also identify problems similar to those identified by Mišura and Žagar [MŽ16]. Current data marketplaces only enable trading of discrete data packages. Further, Cao et al. mention the lack of data trading functions of present data exchange service platforms and IoT platforms. Based on these insights, the authors develop a new design for a data marketplace especially to handle real-time human sensing data that encourages data providers to publish their data.

During the requirements engineering process, the requirements of a data marketplace for real-time human sensing data have been analyzed. These requirements include effective near real-time distribution of data, push and pull communication, data contract management, monitoring quality and an Application Programming Interface (API) for providers and consumers. The result is an architecture consisting of several services. Data providers use the management services in order to offer their data on the data marketplace. A data discovery service provides the means for data consumers to find specific offers. The data is delivered from producers to consumers through data buses. As already has been discussed in [MŽ16], credibility is of major importance. Therefore also the architecture of Cao et al. includes services for data quality analysis and a payment service that monitors the databuses [CPV+16].

Finally, the authors of [CPV+16] provide a comparison of the developed platform to other platforms. In this comparison, it stands out that the majority of platforms for the most part do not provide real-time streaming data. Furthermore the provided solution offers flexible cost and contract models and online billing, whereby the other platforms do not. With a real application scenario (Traffic Information System), the benefits of data marketplaces are again highlighted.

In [KPCK17], the authors also propose a new marketplace platform for the IoT called Intelligent IoT Integrator (I3). The platform provides mechanisms including a user interface for each user group, a registry, a recommendation system, a rating system, mechanisms for billing and payments, real-time data routing, historical data access, authorization, metering, privacy, different data formats and APIs.

The registry stores the important information about all participants of the data marketplaces as well as the offered data streams. The important functionality of matching buyers to data streams is facilitated by a recommendation system. It provides the buyers with a reliable way to identify and find the data which they need. Trust is built by using a rating system, whereby buyers and sellers can rate each other. Billings and payments can be realized in various ways, whereby the system utilizes data access and metering mechanisms to determine the exact price based on the data which got traded. Real-time data routing is realized through a data routing middleware, for example, message brokers where the devices publish generated data on specific topics and authorized data consumers can subscribe to these topics. In some situations, data consumers may not need access to the data in real time, but rather historical data, which is achieved by providing a data repository for historical data [KPCK17].

The previously discussed data marketplaces have many features and problems in common, so it is time to explore new approaches. In doing so, it makes sense to examine decentralized approaches based on blockchain technology in more detail.

3.2 Blockchain-based Data Trading

3.2.1 Blockchain-based Payment Protocols

Lundqvist et al. [LDBA17] introduce a single fee micropayment protocol using blockchain technology. The authors have created a smart cable and socket system as a proof of concept, where the smart cable is able to pay the smart socket for the transmitted electricity. The payments are made without human interaction using the Bitcoin system. This has given the authors the task of developing a protocol that can handle the high fees of Bitcoin transactions. It turned out that one Bitcoin transaction per energy request is far too expensive, in terms of transaction fees. This led to the development of a single fee micropayment protocol. The smart socket retains the signed transaction of the smart cable until a certain limit is reached. It will steadily increase the value of the transaction and have the transaction signed by the smart cable. After reaching the threshold, the smart socket will proceed to broadcast the transaction to the Bitcoin network. The longer

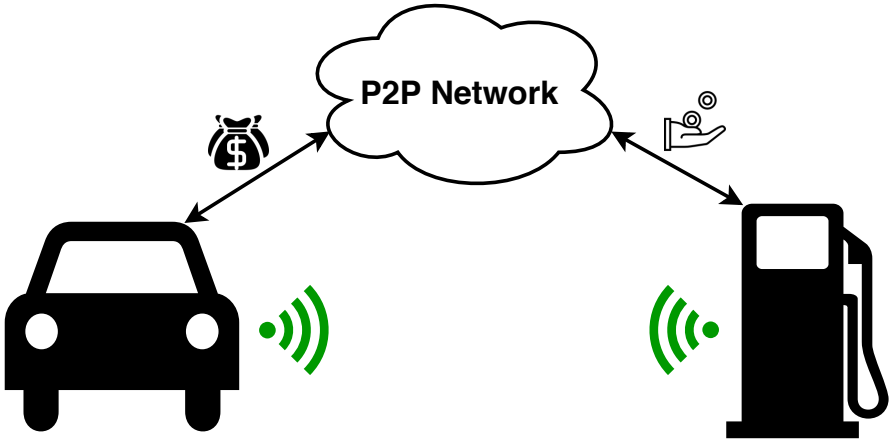

Figure 3.3: Automated gasoline purchases between a vehicle and a gas station with AGasP proposed in [HHL18]

the socket waits to broadcast the transaction, the higher is the risk of double spending. The advantage though is, that only the transaction fees for a single transaction have to be paid. The solution still suffers from the scalability issues of Bitcoin, which make short sessions not worthwhile. Confidence must also be taken into account as the two parties may try to cheat, by not providing a payment or electricity. Standards need to be set, so that it can be widely used [LDBA17]. Data marketplaces also offer the possibility to realize thing-to-thing payments, where things pay for data provided from other things.

Another application of blockchain technology for thing-to-thing payments was proposed in [HHL18]. The authors recognize that blockchain technology enables IoT applications to act transparently in a trustless environment and can help to improve the lack of longevity. In order to study the limitations and possibilities of blockchain technology, Hanada et al. decide to use blockchain technology to realize an application for automated gasoline purchases between a vehicle and a gas station (see Figure 3.3). They provide the design, implementation and an evaluation of this application and further evaluate the usage of smart contracts for machine-to-machine interaction. The application called AGasP uses the Ethereum smart contract platform, where a single contract is deployed on the blockchain and is responsible for the trade. Owners of gas stations publish their products with the corresponding prices using the smart contract. The car must have an account with Ether, which is used to pay for the fuel. The user initiates the payment via the vehicle application. The gas station is able to see the payment on the blockchain and to allow the sender to refuel. The car can be identified by using a short-ranged wireless protocol at the gas station. After refueling, the gas station sends a transaction to the smart contract including the type and amount of fuel, which was dispensed. The smart contract then transfers the correct amount to the petrol station operator and the remaining money to the car. This solution will bind the lifespan of the application to the Ethereum network, removes the need for trust in central payment authorities and ensures

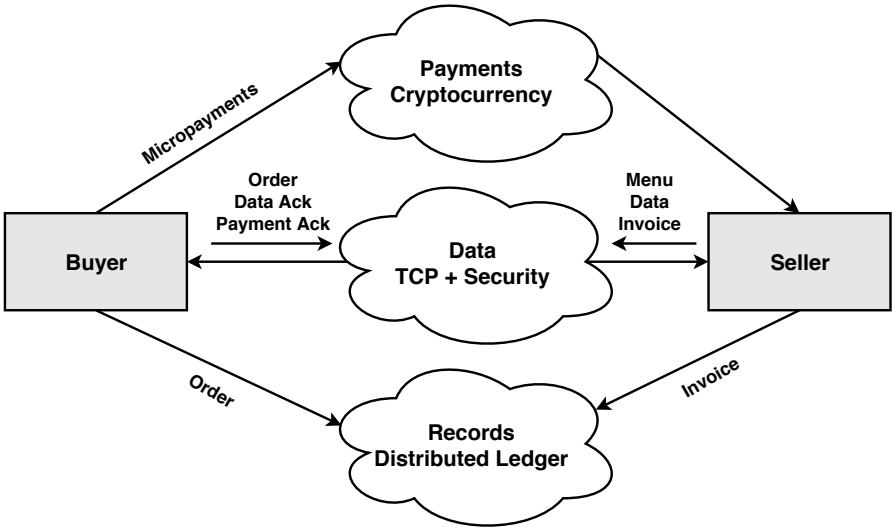

Figure 3.4: Abstract architecture of the Streaming Data Payment Protocol proposed in [RK18]

transparency by logging every transaction on the blockchain. Nevertheless, performance and privacy are only limited using this solution, since the Ethereum network does not provide a high throughput of transactions and everyone is able to see the transactions on the blockchain.

Besides protocols for thing-to-thing payments, data payment protocols also play an important role. The idea of monetized data streams was taken up by the authors of [RK18]. Data consumers should be able to get data streams in exchange for monetary payments. The authors describe a Streaming Data Payment Protocol (SDPP) to realize a micropayment system for IoT data streams. The protocol is based on distributed ledger technology and the reference implementation uses IOTA both as a cryptocurrency and distributed ledger. The protocol basically consists of three different components (see Figure 3.4). These include a data channel, a payment channel and a record medium. The payment channel utilizes the Transmission Control Protocol (TCP) as its underlying transport protocol and for basic security features. For the payment channel, the authors leave open what form of electronic payment is used. However, the records medium must use distributed ledger technology to store the records. Buyer and seller exchange messages in JavaScript Object Notation (JSON) which include a message type, the data, a signature and verification data. The buyer initiates the connection with a hello message, to which the seller is going to reply with a menu that consists of the offers. The buyer can now make an order. After that, the buyer uses the public key of the seller to exchange a symmetric encryption key, which is used for further communication. Consequently, the data flow starts and the buyer pays the seller based on the granularity defined in the menu. Every order, invoice and payment is recorded on the distributed ledger.

Secure Pub-Sub (SPS) is another protocol which is proposed by Zhao et al. [ZLM+18] to enable reliable data distribution in the environment of Cyber Physical Systems (CPSs). The architecture differs from traditional pub-sub solutions in that middleware is not required due to the use of blockchain technology. With this approach, the authors strive to achieve certain properties, including confidentiality, authentication, scalability, completeness, fairness and anonymity. With these properties and possible attack scenarios in mind, Zhao et al. create blockchain-based fair payments with the additional use of a reputation system.

Within the SPS model, data providers advertise on the blockchain which topics they offer. Data consumers announce their interest in certain topics and make a deposit. The provider uses the blockchain system as a communication medium, whereby the data to be transmitted is to be stored in encrypted form on the blockchain. Afterwards, the stored data can be retrieved and decrypted by the consumer. The reputation system ensures that only publishers with a reputation score over a specific threshold are able to publish a transaction [ZLM+18].

The authors of [ZLM+18] state in their comparison of the SPS model with other models that only the SPS model ensures confidentiality, verifiability, anonymity, no trust, fairness and reputation. It should be noted, however, that the used blockchain systems do not grant anonymity, but merely pseudonymity. Furthermore, the scalability of this solution must be considered as well, as the data is stored on the blockchain. Considering the throughput and storage space of Bitcoin or Ethereum, this can become a significant problem.

Fairness is one of the most important points to keep in mind when it comes to data trading. It protects the buyer from buying data that does not comply to certain conditions or is never transferred or revealed. On the other hand, it also protects the seller from buyers that want to obtain the data without paying for it. That is why many works are concerned with ensuring fairness as good as possible. Delgado et al. [DSPSNAHJ17] propose a protocol for this purpose which enables fair data trading based on Bitcoin transactions. The authors make use of private key-locked transactions by providing a new Bitcoin opcode and exploiting an ECDSA vulnerability.

The protocol can be divided into three different phases: the data correctness proof, the signature commitment and the private key exchange. Beginning with the data correctness proof, the buyer receives encrypted chunks of the requested data. The data must ensure a condition which the buyer stated in the request. Based on a cut and choose protocol the buyer selects a random subset from which the seller has to provide a correctness proof based on the condition. With this evidence, the buyer can be sure that the data is correct with a certain probability [DSPSNAHJ17].

Subsequently, the signature commitment phase can be carried out. The buyer requests a signature from the seller including a specific nonce. The seller generates a signature using its secret key and the received nonce and transmits it to the buyer which can in turn verify the signature by using the public key of the seller [DSPSNAHJ17].

In the last part of the protocol, the buyer broadcasts a private key-locked transaction including the payment for the data. After that, the seller can receive the funds by providing a signature with the same nonce as in the preceding phase which is then used by the buyer to restore the private key and decrypt the data [DSPSNAHJ17].

All of the previously discussed protocols take advantage of the blockchain. Consequently, we will also look at different solutions of decentralized data marketplaces, which may use similar protocols to implement data trading.

3.2.2 Blockchain-based Data Marketplaces

Subramanian [Sub18] discusses the limitations that come with company-controlled electronic marketplaces and further describe an alternative approach based on blockchain technology and the advantages of various decentralized electronic marketplaces including marketplaces for digital products. He explained the limitations of company-controlled electronic marketplaces and the benefits of decentralized marketplaces through three key features of a marketplace.

Matching buyers and sellers is one of these key features. Today's company-controlled electronic marketplaces do not match market characteristics efficiently. Single firms can manipulate the market just to protect their own interests without paying attention to the interests of the users. Among other things, the owner would be able determine who is allowed to participate and also certain products selected by the owner could be preferred and better listed. Decentralized marketplaces allow buyers and sellers to independently manage their products and distribute the information in the network. Furthermore, the blockchain can be queried by multiple different services and not only through one interface provided by a single authority [Sub18].

Another key feature of marketplaces are transactions which enable buyers and sellers to exchange assets. Company-controlled electronic marketplaces exhibit certain problems regarding trust, privacy and transaction cost. Trust is ensured through the usage of third-party services or review and rating systems that can not guarantee correct results. In order to be able to carry out a transaction, several external parties are required which results in high transaction cost. Furthermore, a central authority has to control critical data about the buyers and sellers. A decentralized solution on the other hand provides reduced transaction cost because no third parties are involved and provide increased security because the transactions can not be manipulated [Sub18].

Furthermore, issues with regard to institutional infrastructure are also clarified. Traditional electronic marketplaces require contracts that are enforced through the exchange of assets. A decentralized marketplace based on blockchain technology could use smart contracts which provide the means to automatically enforce contracts through the network [Sub18].

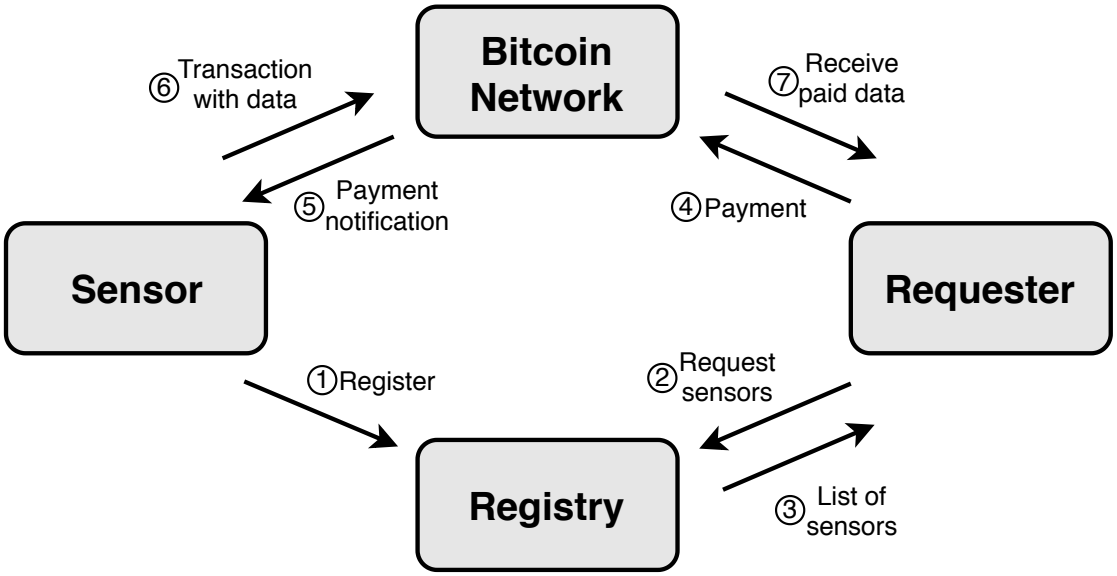

Figure 3.5: Process of exchanging data with Bitcoin by Wörner and Von Bomhard [WvB14]

Wörner and Von Bomhard [WvB14] propose a system, where sensors are able to offer their measurements in exchange for electronic cash. The payment is intended to create an incentive so that the owners of sensors make their data accessible. The authors mention that most sensors are currently only deployed in private sensor networks and are therefore only accessible to a limited number of applications. The goal, however, is to provide a decentralized infrastructure that allows the exchange of data on a world-wide data market.

The system proposed in [WvB14] consists of a sensor client, a requester client and a sensor repository (see Figure 3.5). The sensor client is the provider of the data and has to respond to payments by creating and publishing transactions with the bought data on the blockchain. A requester client wants to consume data and therefore pays the consumer by issuing Bitcoin transactions. The sensors which provide the data are registered in the sensor repository in order to enable requesters to find sensors. Entries of the sensor registry contain basic information about the offered product and additional metadata. Further, the authors provide a prototypical implementation of the system.

However, the system still has a few issues that need to be considered. As the authors have already mentioned, the data is publicly stored on the blockchain, making it freely accessible to third parties. Especially with regard to scalability, storing the data on the blockchain is a big problem, because it would take up a lot of storage space. Furthermore, the transmission of the data is also limited by the throughput of the system.

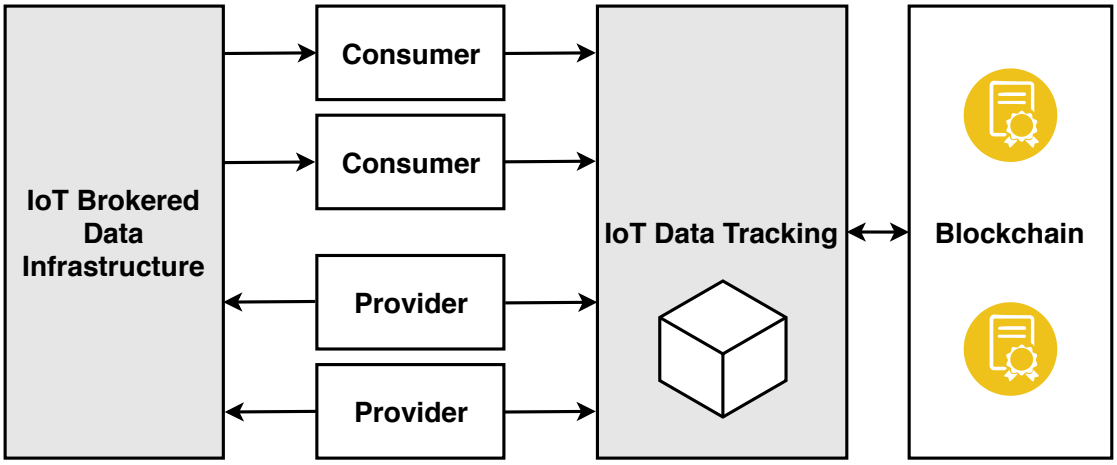

Figure 3.6: Abstract architecture for decentralized metering of IoT data by Missier et al. [MBC+17]

Missier et al. [MBC+17] argue that a data marketplace should not belong to anyone and come up with an architecture for IoT traffic metering and contract compliance (see Figure 3.6). Using blockchain technology, the authors expect to be able to implement a data marketplace that manages itself through the use of smart contracts. They propose a data marketplace with four essential properties. The data marketplace should be flexible and dynamic, should be able to be used by anyone who wishes to earn profit with their data, IoT streams are the main trading asset, and the data marketplace should be run completely decentralized. The architecture is build on top of the IoT brokered data infrastructure, whereby the authors assume that communication is mediated through an infrastructure that consists of message brokers to exchange data between IoT devices. Buyers and sellers agree upon specific topics, which will be accessible for the buyers for a certain timeframe. Contract enforcement and settlement is realized through traffic cubes, wherein all or partial data flows are recorded. Producers and consumers only construct unilateral traffic cubes, since the broker decouples the communication between producer and consumer, whereby the broker has the whole view of all data flows. The cubes are published through transactions on the blockchain to ensure transparency and checked for consistency for the settlement. The authors further provide a proof of concept using the Ethereum platform, the open-source Mosquitto Message Queuing Telemetry Transport (MQTT) broker and Apache Cassandra to store the traffic cubes [MBC+17].

The design of [MBC+17] focuses on decentralized metering of IoT data and trustless contract settlement which of course is an important part of a decentralized IoT data marketplace. Nevertheless, there is more to a data marketplace than just traffic metering and contract settlement. A data marketplace also needs to provide a platform to discover and select certain products. It is not specified how consumers can find a suitable topic which provides the desired product. For some participants, historical data may also be of interest. Unfortunately, this solution only considers the sale of real-time streaming data.

In addition, the incentive that a third party should operate a broker is missing. The broker should be rewarded in any form for the services offered, so everyone is tempted to operate a broker and participate on the data marketplace. Therefore, it is necessary to consider those additional aspects in future works.

Another data marketplace based on blockchain technology is proposed by Banerjee and Ruj [BR18] which mention that current data marketplaces are not suitable as they fail several essential properties. The authors describe a blockchain enabled data marketplace that fulfills fairness, efficiency, security, privacy and adherence to regulations. In the design of the data marketplace, the blockchain is used as a trusted third party, which ensures these properties. For a fair data trade, the participants have to agree upon a price and the commodity. The buyer must receive data that adheres to the previously defined conditions and the seller has to get the right amount of money. The design of the data marketplace ensures fairness by using a FairSwap protocol, where buyer and seller agree over a certain condition expressed as a boolean circuit. A judge contract is responsible for managing the funds until the buyer agrees to the trade or raises a valid complaint. Transparency, security and privacy are given through the usage of the distributed public ledger, whereby the data is encrypted and can be verified using zero knowledge proofs or proofs of misbehavior. Different regulations for data are also described by a predicate to verify if the regulations are followed. The authors also provide suggestions to efficiently define and verify the condition and regulation predicates, but mention it as a difficult hurdle to overcome.

While Banerjee and Ruj [BR18] provide an interesting approach to implement fair data trading using blockchain technology, they neglect many other important aspects that are needed to implement a data marketplace. Although buyers, sellers and mediators need to register at the data marketplace, they do not determine how they can discover other participants or data products. Furthermore, no other features like price negotiation are offered, since the authors have focused exclusively on data trading. Among other things, the concept is also not designed for use in the area of the IoT. The IoT imposes different requirements on the design of a data marketplace, which have to be considered. An example of this is the sale of real-time streaming data, which is not discussed in this concept. However, in the IoT, real-time streaming data plays an important role. The authors themselves say that the design is still under construction and has yet to be implemented.

If we not only look at solutions based on blockchain technology, but also solutions based on other data structures, it is especially interesting to take a closer look at IOTA. The cryptocurrency IOTA is specifically designed for the IoT industry [Pop17]. Unlike other cryptocurrencies like Bitcoin and Ethereum, it does not use a blockchain as its underlying data structure. Instead, IOTA uses a directed acyclic graph, which is called the tangle (see Figure 3.7). Transactions are stored in the tangle and to add a new transaction, the sender has to validate two other transactions. In 2017, the IOTA Foundation launched a data marketplace based upon IOTA. In the announcement [Sø17], the IOTA Foundation refers to data as one of the most important parts of future machine

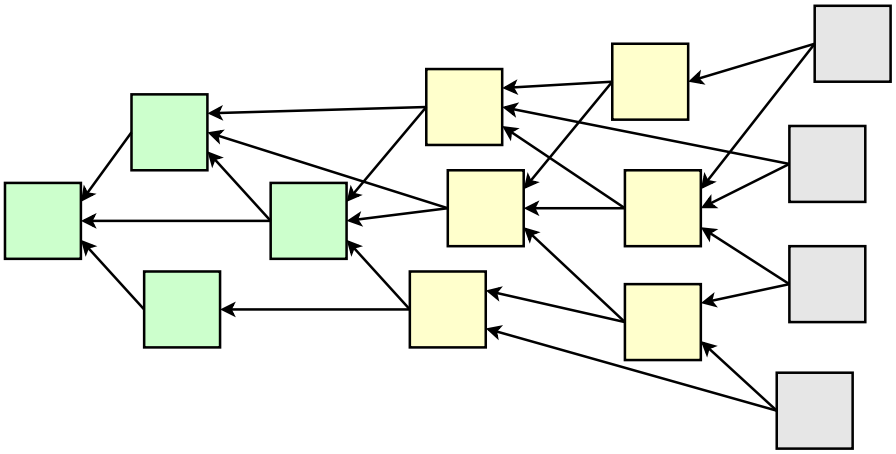

Figure 3.7: A directed acyclic graph (the tangle) in which all transactions are stored

economy where billions of different devices will trade various commodities. As it has already been mentioned in the discussion of other papers, they also see the waste of data which end up in data silos as the biggest problem.

This is where the solution of a open and decentralized data marketplace could help to use this wasted potential. The marketplace developed by the IOTA foundation uses the tangle as a secure data communication protocol and zero-fee microtransaction system [Har19]. As the IOTA foundation also tries to create such a decentralized data marketplace, it is particularly interesting to look at their approach, to identify differences between the tangle-based model used by IOTA and the blockchain-based model envisioned within the thesis at hand. IOTA, however, still has to deal with the problem that it depends on a central component in the system. Currently, they are working on a solution to solve this problem that focuses on node responsibility, automatic peering, rate control and innovation of the consensus model [CT19].

Coming back to the discussion of blockchain-based approaches, Gupta et al. [GKJ18] propose a three-tier framework (see Figure 3.8) for a dynamic and decentralized IoT data marketplace. The authors also recognize the huge potential of peer-to-peer communication and blockchain technology to realize a data marketplace with a decentralized architecture. The intention behind this approach is to overcome the disadvantages of existing solutions based on a centralized architecture. Centralized solutions tend to suffer from scalability issues, expensive infrastructure and usually have a single point of failure. Gupta et. al. [GKJ18] mention that other blockchain-based solutions failed to pay attention on important components like discovery, contract creation, settlements and reliability and thus recognized the need for a more elaborate design.

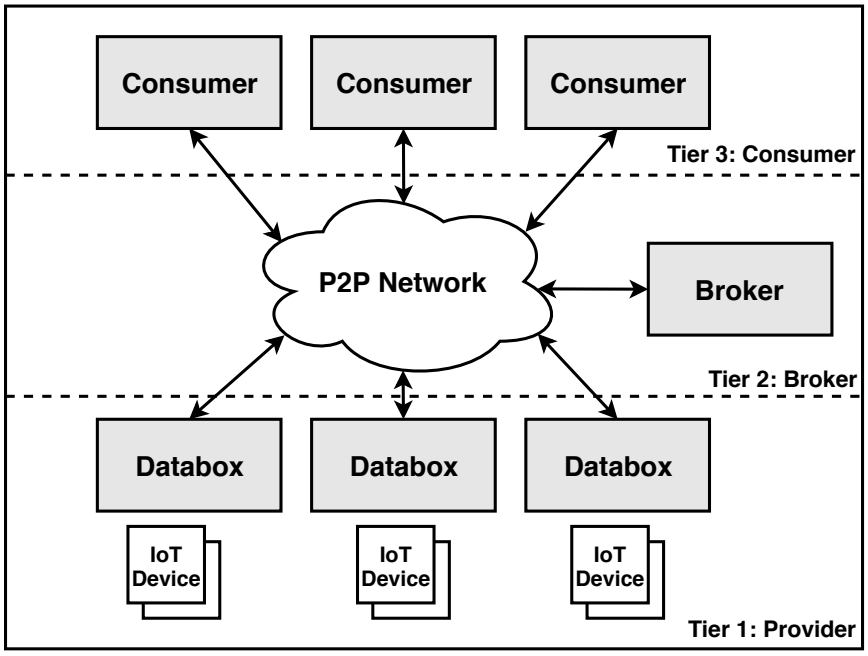

Figure 3.8: IoT marketplace framework by Gupta et al. [GKJ18]

The proposed framework consists of three major participants, including data providers, data consumers and brokers. Providers and consumers perform the same role as in most other works about data marketplaces, with the provider supplying the data from IoT devices and the consumer buying the published data of interest. The role of the broker is inherited from a highly-resourced device. Its task is to mediate the data trading process, by providing registration, discovery and selection functionality. The broker is responsible for matching buyers with sellers and is encouraged by a fee-based system [GKJ18].

As has already been mentioned earlier the framework relies on blockchain technology. Data trading is based on the usage of smart contracts, whereby the participants are able to negotiate with each other before proceeding with the contract. Further, the solution also incorporates a reputation framework, encrypted data transmissions and a settlement solution that checks the quality of the trade. This includes the verification of the data transfer counters of the participants which is also used in similar ways by other solutions. However, the design needs to consider further important aspects, including privacy, security and malicious behaviour of participants, as the authors have already stated [GKJ18].

Also, Ramachandran et al. [RRK18] examine how blockchain technology can be used to create a decentralized data marketplace for smart cities. The authors mention that smart cities are driven by data-intensive applications which require a high number of IoT devices. A data marketplace would provide participants the possibility of gaining economic benefit from their generated data as well as the possibility to buy and find

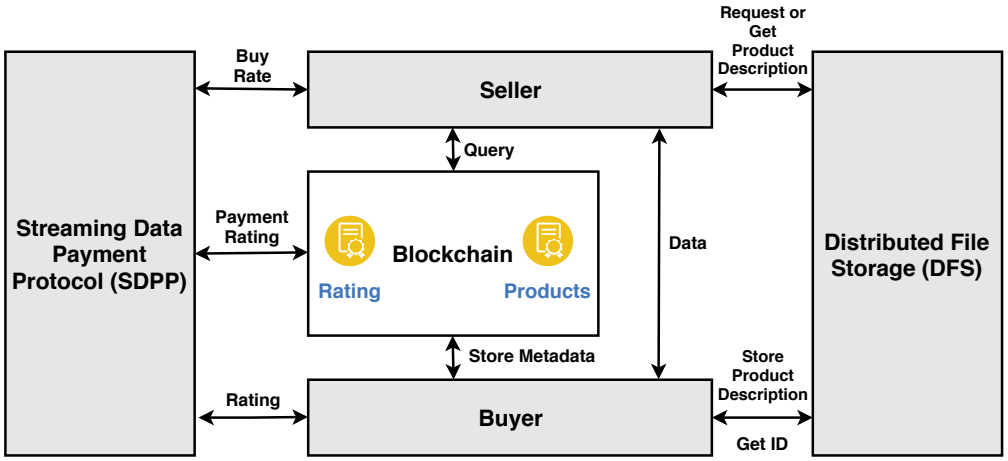

Figure 3.9: Abstract overview of the decentralized IoT data marketplace by Ramachandran et al. [RRK18]

data for their own interests. This data can then be used to improve the quality of lives in smart cities. One example of real-time IoT data marketplaces discussed in [RRK18] is I3 (see Section 3.1).

However, since I3 is a central data marketplace, the authors of [RRK18] argue that they are developing a decentralized data marketplace. One major reason for this is to prevent operators of centralized data marketplaces to misuse their monopoly market power to direct the market. Furthermore, blockchain technology would be able to facilitate trust and transparency and the design of a decentralized data marketplace could help to tackle the problem of market inefficiencies which are often inherent to centralized solutions.

The authors of [RRK18] define several key components of a data marketplace. There must be buyers and sellers in the smart city which want to provide their data and are willing to buy and consume data. Furthermore, a discovery mechanism must be provided that enables buyers and sellers to find each other. Another very important part is the organization of metadata which are provided by the seller. The metadata is responsible for describing the offered data streams and especially relevant for the discovery mechanism. Sellers want to make economic benefit from providing their sensor data and buyers want to receive the bought data. Therefore, streaming data and payment protocols like the SDPP (see Section 3.2.1), cryptocurrencies and other infrastructure for reliable data distribution must be considered. Furthermore, data quality, trust, security and privacy are major key components, as we know from other works which have already been discussed in this chapter.

For the implementation of the decentralized data marketplace (see Figure 3.9), the authors make use of the InterPlanetary File System (IPFS) which is a Distributed File Storage (DFS) to store the product information in JSON format. The metadata is saved on the blockchain and the buyer makes use of both the blockchain and the DFS to find

the desired product. The implementation uses SDPP and after every trade buyer and seller can rate each other. Smart contracts on the Ethereum blockchain are used for the registration of the products as well as the rating [RRK18].

In contrast to other works which have been discussed earlier, the authors of [RRK18] make an effort to consider other functionalities that are needed by a data marketplace besides data trading. However, one thing that is not considered are price negotiations. Participants of the data marketplace are not able to negotiate about the price of the data products. As a result, buyers are forced to pay a fixed price. Another important aspect that is not considered is how the IoT data is integrated in the system. If the buyer or seller is an IoT device, it must be considered that they have only limited resources. The buyers and sellers are communicating directly with each other using the SDPP, thus they must be able to handle multiple connections and perform cryptographic operations.

3.3 Conclusion

In this chapter, we discussed related work on data marketplaces. We started with solutions that are based on a centralized approach and then moved on to solutions that use blockchain technology to explore the potential of blockchain-based payment protocols and decentralized data marketplaces.

The discussion about traditional data trading included the SaaS model and different solutions for centralized data marketplaces. As a result, it was possible to gain some knowledge about the requirements which are placed on IoT data marketplaces. Although these solutions give a first insight into the topic, they are not suitable for use in the area of the IoT, as we have already noticed. We were able to identify several disadvantages of centralized data marketplaces. These include scalability issues, expensive infrastructure, single point of failures, trust problems and privacy issues which led to the exploration of blockchain technology in this research area.

Various works about blockchain-based data trading protocols have been discussed and provided a short introduction on how data trading can be implemented using blockchain technology, which is also an essential part of decentralized data marketplaces. Several advantages regarding transparency, security, trust and scalability have been identified, which have emerged through the use of blockchain technology. Consequently, some work has been discussed on decentralized data marketplaces, which also benefit from the application of blockchain technology. It turned out that the use of blockchain technology brings great potential to create a decentralized data marketplace for the IoT. However, any related work must also be viewed critically in order to be able to identify good solutions which could provide practicable ideas or problems which have to be addressed in future works.

In the course of the discussion about blockchain-based data marketplaces we where also able to identify several problems of already proposed solutions on decentralized data marketplaces. The main problem here is that most solutions do not provide essential

key elements which are needed by a data marketplace but rather focus primarily on data trading. Although this is the main element of a data marketplace, it is not possible to operate a data marketplace that does not pay attention on important other key elements. For example, Missier et al. [MBC+17] focused primarily on decentralized metering of IoT data and contract settlement. Also the work of Banerjee and Ruj [BR18] does not cover all important functionalities of a data marketplace. Nevertheless, these two works offer approaches that can be helpful for the development of a decentralized data marketplace and therefore should be considered more closely.

The problem of missing key elements was also addressed in the work of Gupta et al [RRK18]. That is why the authors not only deal with data trading but also with product information, discovery and selection, contract creation, settlements, reliability, ratings and price negotiations. Also in [RRK18] the authors describe multiple key components of a data marketplace including query and search, product and seller management, rating, curation and recommendation, secure identity verification, data transfers and payments. Compared to the works which have been mentioned in the last paragraph, these works consider a much broader range of key elements. The consideration of different key elements is very important, because only the interplay of these enable a data marketplace.

Nevertheless, there are ways to improve the already proposed solutions regarding fairness, malicious behaviour of trustless nodes, incentivation, data integration and delivery, just to name a few examples.

CHAPTER 4

Smart Contract and Decentralized Application Platforms

In this chapter, we will take a closer look at different smart contract and decentralized application platforms. The main goal is to find a suitable platform that can later be used for the implementation, to realize the trading of data based on the use of smart contracts. For that, we will discuss several smart contract and decentralized application platforms and then compare them. The comparison (see Section 4.5) consists of nine different characteristics, which include smart contracts, programming languages, structure, openness, consensus mechanism, membership selection, scalability, energy consumption and finally support and tools. The chosen platforms show significant differences in most of these points. The comparison will then later serve as a decision-making aid in the implementation phase in order to use the appropriate platform. In the beginning Bitcoin (see Section 4.1) will provide a good introduction as it is the first blockchain-based cryptocurrency ever created and is already making its steps towards smart contracts, but is very limited. After that, Ethereum (see Section 4.2), which has already been introduced as a smart contract and decentralized application platform, is further examined and represents the first possibility to create decentralized applications. Later we will look at Neo (see Section 4.3), which strives to enable smart economy, by providing essential building blocks of which one are smart contracts on the Neo blockchain. The last platform discussed is Hyperledger Fabric (see Section 4.4), a distributed operating system for general purpose blockchains. Of course, there are many more platforms that could be discussed, but Bitcoin, Ethereum, Neo and Hyperledger Fabric should suffice for the decision making process, since these platforms are among the most relevant in today's blockchain landscape.

4.1 Bitcoin

Bitcoin is a peer-to-peer electronic cash system introduced by Satoshi Nakamoto [N+08]. Bitcoin is not necessary a smart contract and decentralized application platform, but was the first introduction of blockchain technology. Nevertheless, it provides scripting on the blockchain, which can be seen as constrained smart contracts. This will provide the possibility to see the limitations of Bitcoin compared to smart contract and decentralized application platforms like Ethereum and others.

4.1.1 Blocks and Transactions

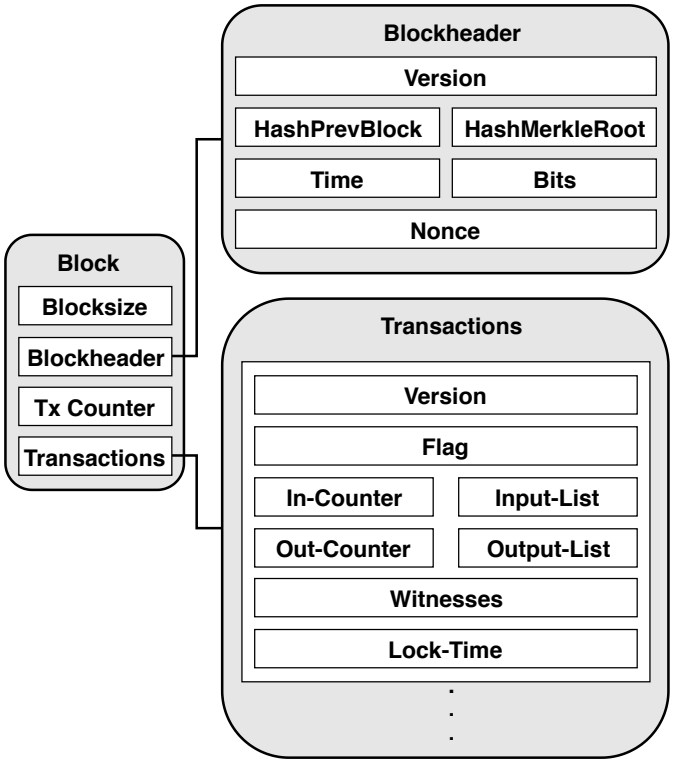

Figure 4.1: Block and transaction structure in the Bitcoin system

The blockchain is the underlying data structure of Bitcoin, which represents the distributed append-only ledger, in which all transactions are stored. The transactions are not stored individually in the blockchain but grouped in blocks with a maximum size of one megabyte. The blocks (see Figure 4.1) contain the actual size of the block, the header, a transaction counter and a list of all transactions which are part of the block. The block header contains a version number, the hash of the previous block in the chain and a timestamp. In addition, it contains important fields which are necessary for the mining process, the difficulty (bits) and the nonce. Furthermore, it also contains the hash of the Merkle tree of all transactions in the block. By constructing a Merkle tree with the transactions,

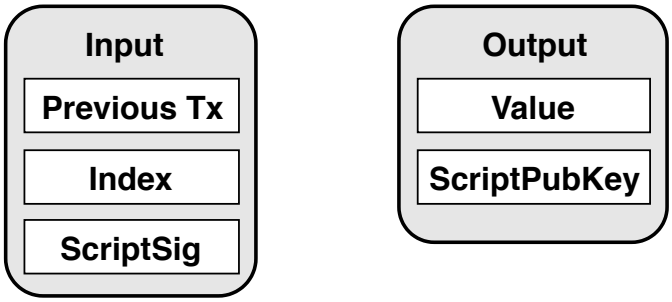

Figure 4.2: Input and output structure of a transaction in the Bitcoin system

the integrity of the transaction set can be verified very efficiently. Further, a Merkle tree provides a very efficient method to check if a transaction is included in the tree, because it only needs $2 * log_2(n)$ calculations [Ant14]. In Bitcoin, a coin is represented by a series of transactions. A transaction consists of a version number, a flag, the inputs and outputs (see Figure 4.2), witnesses and the lock-time. The inputs of a transaction point to older transactions in the blockchain, which have unused outputs. These outputs are referred to as Unspent Transaction Outputs (UTXOs). Inputs and outputs contain additional fields called scriptSig and scriptPubKey, which combined form a script that has to be executed by the node. Generally, it is used to determine if the sender of the transaction is allowed to spend the outputs.

4.1.2 Mining and Consensus

Every participant in the Bitcoin network has to verify new transactions. The participants include lightweight nodes, which only verify new transactions based on Simplified Payment Verification (SPV). These nodes only have to download block headers instead of the whole block [Ant14]. Full nodes store the entire blockchain and are also responsible for verifying new blocks. The process of creating a new block is called mining. A mining node in the Bitcoin system has several tasks. A miner has to maintain the blockchain, receive and verify new transactions or blocks and create new blocks. To create a new block, a miner must perform the PoW. Earlier on, we have already discussed the property puzzle friendliness of cryptographic hash functions, which plays a major role in the PoW algorithm. The PoW is a hash puzzle, where the miner has to find a nonce, such that the hash of the block including the nonce falls in a specific target space which is specified by the difficulty. In the case of Bitcoin, a miner has to compute hashes based on SHA-256. The difficulty is used to react on the changing computing power of the whole network. Many miners are competing to create a new block. This process requires a tremendous amount of processing power to get a realistic chance. That is why many smaller miners join mining pools to combine their mining power to search for a solution together. If the mining pool finds a new block, which gets included in the blockchain, the involved miners will split the reward based on certain criteria. Since nobody would provide their computing power for free, miners will get a reward for successfully creating

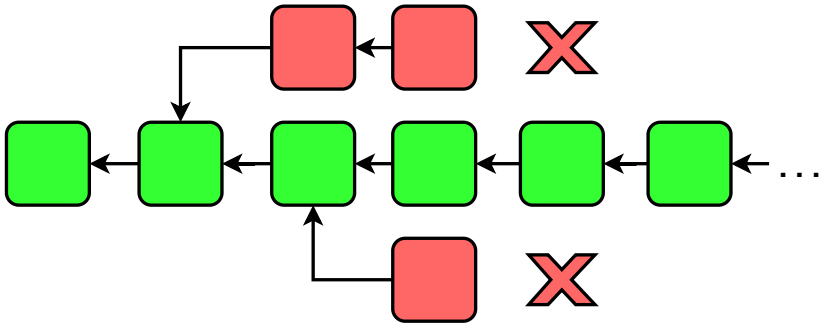

Figure 4.3: Forks in the Bitcoin blockchain

a new block. The reward is created by including a so-called coinbase transaction in the block, which creates new coins. The miner can specify the new owner of the coins. The reward for creating a new block started with 50 Bitcoins and is halved after a certain amount of blocks got created, until no reward is left. Miners additionally get to collect transaction fees. The transaction fees are the difference between input and output value of a transaction. A miner can decide to prioritize transactions based on the provided transaction fee.

Bitcoin is a peer-to-peer system in which everyone is allowed to join the network and also to participate in the mining process. All nodes have to reach consensus over the next block, which should be included in the blockchain. Because each node creates new blocks independently of the other nodes, there may be an inconsistent state of the blockchain throughout the whole system, since not every node sees the same blocks at the same time. This leads to forks in the blockchain (see Figure 4.3). The participants will decide based on the rule to build upon the longest chain on which fork a new block should be attached. Other blocks will get orphaned and thus the transactions will not be included in the main chain. Transactions that are already buried deeper in the blockchain are therefore more likely to be in the longest chain and the probability that the transaction will stay in the longest branch is higher.

Bitcoin suffers a severe scalability problem. With the restriction of only one megabyte per block and an average transaction size of 250 bytes, a block is only able to contain 4000 transactions. Further, a new block is only created about every ten minutes, which limits the throughput of Bitcoin to 7 transactions per second. For comparison, the payment network Visa has already peaked with 47 000 transactions per second. On average Visa should be able to handle 2 000 to 10 000 transactions per second. Besides Visa also Paypal can handle up to 100 transaction per second [NBF+16].

Changing the protocol is not a trivial task, thus changing the block size would require a soft or hard fork of the blockchain. Further implications of changing the block size also have to be considered. For example the bandwidth and resources needed to run a full node. This could lead to a more centralized system, since not everyone has the opportunity to run full nodes with more resources [NBF+16].

One possible solution to cope with the scalability problem has been introduced with the Bitcoin Lightning network. This network describes a network of micropayment channels, which are able to scale to billions of transactions per day [PD16]. Using the lightning protocol, only two transactions will be stored in the blockchain. Two parties which are willing to exchange Bitcoins have to open a payment channel. This results in the first transaction, which is stored on the blockchain. Both parties have to send Bitcoins to a multisignature address to fund the payment channel. This balance will be used to transfer balances and thus should be greater than the value which the parties wish to exchange. Due to the nature of multisignature transactions, both parties will have to sign the transaction to spend money from these funds. Both parties are able to exchange transactions, which are not stored on the blockchain. The second transaction which is stored on the blockchain, will send the final amount to the respective parties. It is possible to use existing payment channels and intermediaries are able to issue the transaction. The intermediary can verify that the payment was issued by providing a secret key, which was exchanged by both parties prior to the payment process. This solution allows a much better scalability of the system, nevertheless also has to deal with some problems. The two transactions which are stored on the blockchain will still be subject to the slow throughput and payment channels are also only useful for transactions with a high enough value transfer [CMVM18].

4.1.3 Scripting

Bitcoin uses a Forth-like scripting language, which is simply called Script and has native support for cryptographic operations. It is a stack-based language and not Turing-complete. There are no possibilities for loops, which introduces an upper-bound for memory and computation time. Inputs and outputs of a Bitcoin transaction each include a part of a script, which is used to enforce the conditions that have to be met to transfer the Bitcoins. Bitcoin scripts can be viewed as smart contracts, because every node in the network has to execute the script to verify the corresponding transaction [NBF+16]. Scripts in Bitcoin are stateless and fully self-contained, so every node executing the script will produce the same output. In its simplest form, a script checks if the public key of the receiver matches the digital signature. Hereby, the public key is directly contained in the script and the owner has to present the corresponding digital signature for the private key to unlock the coins. The transactions which are most used are called Pay-to-Public-Key-Hash (P2PKH) transactions and will lock the transactions outputs for a specific public key hash. The output can only be used by presenting the public key and a signature with the corresponding private key. Another important form are multisignature transactions. The outputs of these transactions can only be unlocked by providing a certain number of digital signatures for the public keys contained in the transaction. Bitcoin allows different schemes that can be implemented [Ant14]. It provides a powerful tool to create different schemes how inputs and outputs can be used. Nevertheless, it can not be compared to a smart contract and decentralized application platform which allows arbitrary code execution on the blockchain. Statelessness and Turing-incompleteness are very restrictive and hinder the usage of Script in a more advanced way.

4.2 Ethereum

Vitalik Buterin proposed with Ethereum a Next-Generation Smart Contract and Decentralized Application Platform [But13]. Ethereum provides a Turing-complete programming language, which gives the possibility to execute code directly on the blockchain. Gavin Wood mentions in [Woo14] that Ethereum can be viewed as a transaction based state machine. The genesis block represents the initial state and by incrementally executing transactions, state changes occur. Unlike Bitcoin, where the state of the system consists mainly of UTXOs, Ethereum can store any information that can be represented by a computer. Smart contracts enable anyone to create decentralized applications that represent state transition rules of the system.

Ethereum overcomes with smart contracts a lot of important limitations of the Bitcoin scripting language. The lack of Turing-completeness is a great constraint of the Bitcoin scripting language. As has already been mentioned earlier, Script does not support loops, which makes code very space inefficient, because the code has to be repeated to simulate a loop. Another limitation is the value-blindness, whereby Bitcoin scripts do not have fine-grained control over the value, which should be transfered. The lack of state restricts it even further, because an UTXO can only be spent or unspent which limits the possibilities for multi-stage contracts. Blockchain-blindness further limits Bitcoin scripts in a way, that they are not able to access blockchain data. Ethereum offers the possibility to create decentralized applications by providing more power than Bitcoin scripting, through Turing-completeness, value-awareness and state [But13].

4.2.1 Global World State

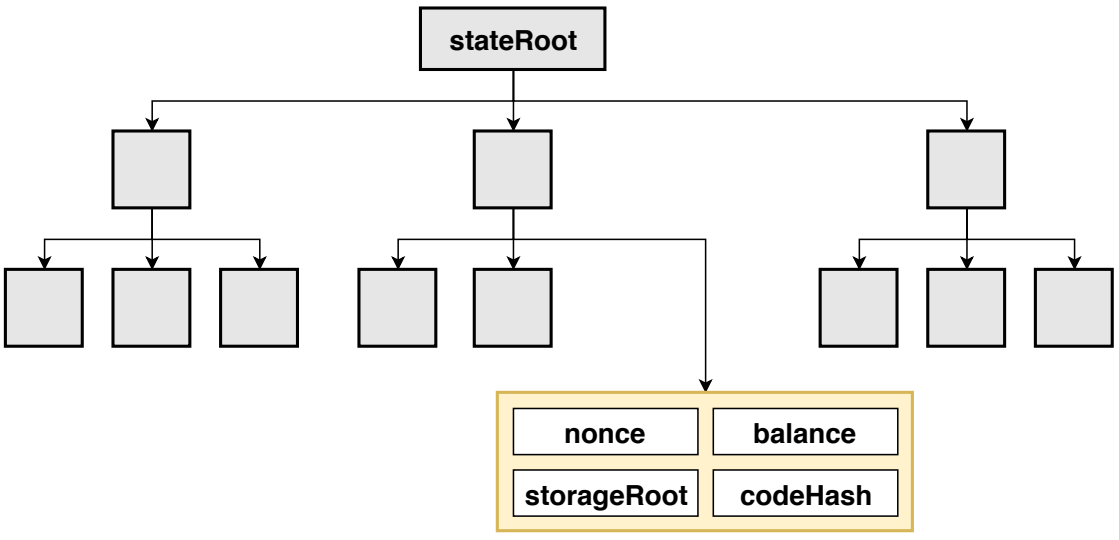

Figure 4.4: Global state trie of Ethereum

The world state of Ethereum consists of accounts that are stored in a modified Merkle Patricia tree (see Figure 4.4), which cryptographically secures the whole state. The world state is not stored directly on the blockchain, but in a separate database backend. On the blockchain, only the root hashes are stored. Accounts have a 20-byte address and state transitions describe the exchange of value or information between accounts. An account consists of four fields: nonce, balance, storageRoot and codeHash. A simple counter of sent transactions or contract creations of an account is stored in the nonce. The balance indicates how many Wei the account has available. The storage of an account is encoded in a Merkle Patricia tree and the root of the tree is stored in the storageRoot field [But13, Woo14]. We can further distinguish between externally-owned accounts and contract accounts. The difference between these types is, that externally-owned accounts are controlled by private keys, whereby the latter are only controlled by their code [But13].

4.2.2 Blockchain and Consensus

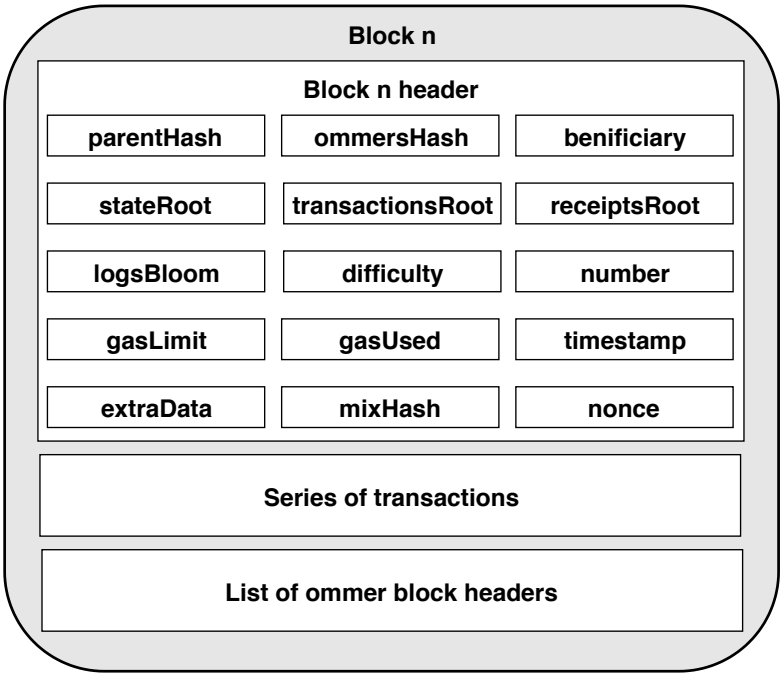

Figure 4.5: The structure of a block in Ethereum

Like Bitcoin, Ethereum also uses a PoW algorithm to achieve decentralization. The PoW algorithm used is called Ethash. The exact specification can be found in [Woo14]. Ethereum's block structure (see Figure 4.5) differs in some ways from other common blockchain structures. Because Ethereum is a transaction-based state machine, the blockchain also has to store the root of the state trie. It is important to mention that only the root is actually stored in the blockchain and not the whole state. Every block

will refer to the most recent state. If the state changes, only a part of the tree has to be changed, because Ethereum uses modified Merkle Patricia tries, which also allow to insert and delete nodes. In addition to the usual checks, which are also made in Bitcoin to check if a block is valid, the state transition function has to be executed for each transaction [But13]. Furthermore, in Ethereum also stale blocks are included in the blockchain. This got implemented to reduce security risks introduced through lower block creation times (15 seconds). Ethereum implements a modified version of the Greedy Heaviest-Observed Sub-Tree (GHOST) rule proposed by Sompolinsky and Zohar [SZ15], which describes the inclusion of stale blocks in the blockchain. The network will not build upon the longest branch of the blockchain, but includes the weight of the subtree in the decision.

The Ethereum Network is set to switch from PoW to PoS in the near future [Pea19]. Consensus through PoW is based on rewards and requires large amounts of computing power. PoW requires high levels of electricity and is more prone to centralization. Another problem is that attackers are not discouraged to attack again. With PoS on the other hand, there is no need for large amounts of energy. 51% attacks on a PoS system are far more expensive than on a PoW system, since PoS offers the ability to apply economic penalties on misbehaving participants [But16]. Buterin and Griffith introduced Casper [BG17], an overlay atop a proposal mechanism, which finalizes blocks. It is a hybrid PoW/PoS consensus mechanism, whereby the PoW algorithm selects new blocks, which results in a tree of blocks. The PoS mechanism for Casper is following the Byzantine Fault Tolerant (BFT) approach, where 2/3 of the network participants have to behave honestly to finalize blocks. It is responsible for the finalization of blocks from the tree previously created by the PoW mechanism. This will create a list of blocks. Because it is very inefficient to look at the whole tree, Casper will only look at a checkpoint tree, where every hundredth block will form a checkpoint. The selection of a unique chain is made by validators, which deposit a certain amount. It is not possible to finalize two conflicting blocks as long as no more than 1/3 of the validators are corrupted. Furthermore, a new checkpoint can always be finalized, as long as more than 2/3 of the validators behave honestly. Casper, unlike BFT algorithms, also punishes misbehaving validators by keeping their entire deposit. It further enables dynamic validator sets and protection against long range revision attacks and catastrophic crashes. In the future, the selection mechanism will be replaced, because PoW is too inefficient.

4.2.3 Transactions and Messages

Ethereum distinguishes between transactions and messages. The differences between these two are, that transactions are always sent from an external actor, whereby a message is always sent by a contract. In their structure (see Figure 4.6), however, both types differ only slightly from each other. First and foremost, a transaction contains the typical fields needed for a cryptocurrency. These include the recipient, the signature of the sender and the amount of Ether to be transfered. Ethereum uses recoverable ECDSA signatures, hence the three signature components v, r and s in the transaction. A transaction also

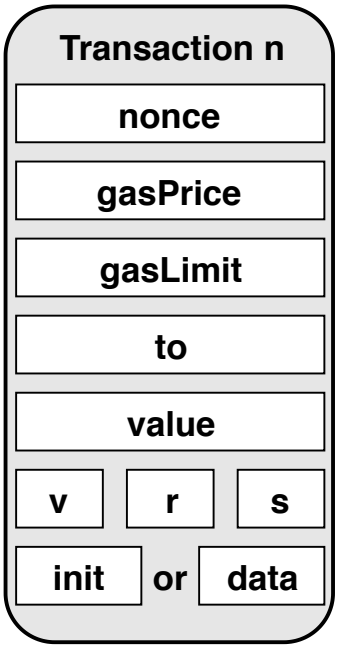

Figure 4.6: The structure of a transaction in Ethereum

includes an optional data field or init field which is an unlimited size byte array that is used for message calls or the account initialisation procedure [But13, Woo14].

For further explanation of the remaining fields of a transaction, it is essential to understand the concept of Gas and Payment in the Ethereum system. All computations in Ethereum require the payment of fees in order to prevent network abuse, infinite loops or other actions which waste resources. Each computational step requires a certain amount of gas. In [Woo14], the fee schedule defines the amount of gas for each opcode. A transaction defines a STARTGAS field to indicate the maximum number of computation steps and the GASPRICE, which defines the price a sender is willing to pay for one unit of gas. The STARTGAS is also called gasLimit, since unused gas gets refunded [But13, Woo14].

4.2.4 Execution Model

Messages are used for the communication between contract accounts. Every time a contract executes the CALL opcode, a message gets generated which results in the execution of the called contract. Messages implicitly contain the message of the sender, the recipient and the amount of Ether to be transfered. In addition, a message also has an optional data field and the STARTGAS field. Like before with transactions, the STARTGAS field helps against malicious behaviour, for example to prevent endless recursive calls between two contracts [But13].

The execution of transactions in Ethereum is described by the state transition function. The exact definition of the state transition function can be found in [Woo14]. The state transition function basically starts with verifying the transaction and then checks if the sender has enough balance in its account to pay for gas and transaction fees. After that, it will deduct the transaction fees from the sender's balance. The gas will get initialized and then the main part of the transaction execution will begin. In case of a value transfer, the state transition function deducts the amount from the sender's balance and credits it to the receiver's account. If the receiver's account is a contract, then code execution starts until it reaches the end of the code or gas is depleted. In the case of an error, the transaction will be reverted, by reverting all state changes and refunding the remaining gas to the sender. Nevertheless, fees have to be paid to compensate the miner for the already used resources [But13].

The execution model of Ethereum is specified by the Ethereum Virtual Machine (EVM). The EVM is a virtual state machine, which is quasi-Turing complete. It is only quasi-Turing complete, because as has already been discussed earlier, the computations are bound through gas. The EVM is specifically designed to support the Keccak-256 hash scheme and elliptic-curve computations, as can be seen from the chosen word size of 256 bit [Woo14]. During its execution, the code has access to three different storage areas. The code can access its own long-term storage, the memory and the stack. The computational state of the EVM can be defined with the tuple (block_state, transaction, message, code, memory, stack, pc, gas), where each instruction effects the tuple in a predefined way [But13].

4.3 Neo

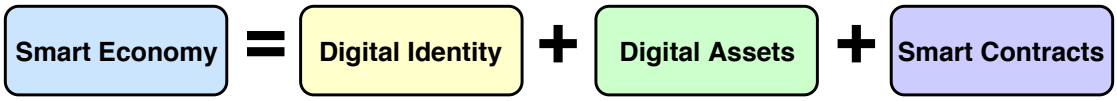

Figure 4.7: The building blocks of smart economy

The Neo network [neo14] is a distributed network especially created to enable smart economy. Neo defines three essential building blocks, which are needed to achieve smart economy. These building blocks include digital assets, digital identity and smart contracts (see Figure 4.7). Physical assets can easily be linked to digital assets by using digital identities. Digital identities act as the link between the virtual and the real world. These identities are implemented based on the X.509 standard [MAM+99]. Neo wants to provide multiple different schemes for the issuance and verification of digital identities. These include the usage of facial features, fingerprints, speech recognition, SMS and other multi-factor authentication methods. To bring all building blocks together, smart contracts will provide the means to automatically manage digital assets. Neo has two different tokens, each with its own utility in the network. The Neo Token can be compared to a company's shares and the number is fixed to 100 million tokens. Neo

holders get management rights of the blockchain, which means they will take part in the voting process to decide the consensus nodes, which carry out the creation of new blocks. The second token in the Neo system is called GAS and is generated with each new block, until the maximum of 100 million GAS is reached. Each block will initially create eight GAS, whereby this amount will decrease annually by one GAS. GAS tokens are necessary to deploy and execute smart contracts on the Neo blockchain. Neo currently uses ECDSA but is working on implementing lattice-based cryptography to protect itself from quantum computers, as ECDSA is not able to resists quantum computers [Ber09].

4.3.1 Blockchain

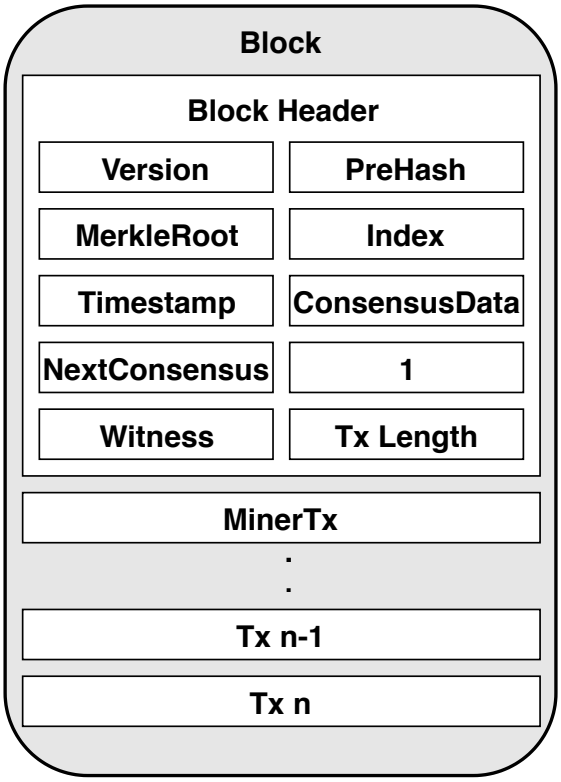

Figure 4.8: The structure of a block in the Neo blockchain

Neo is also based on a blockchain, whereby each block has a block header and a body (see Figure 4.8). The block header includes basic information and verification data, which is necessary to verify the integrity of the blockchain. Since Neo provides single block finality, there will be no forks in the blockchain. The reason for this is the consensus mechanism, which will be explained later on. The structure of the block does not differ greatly from other blockchains except for the fields needed for the consensus mechanism. Typical fields which are also found in other blockchains are the previous hash, timestamp and the Merkle root. Later on, we come to the fact that the nodes which take part in

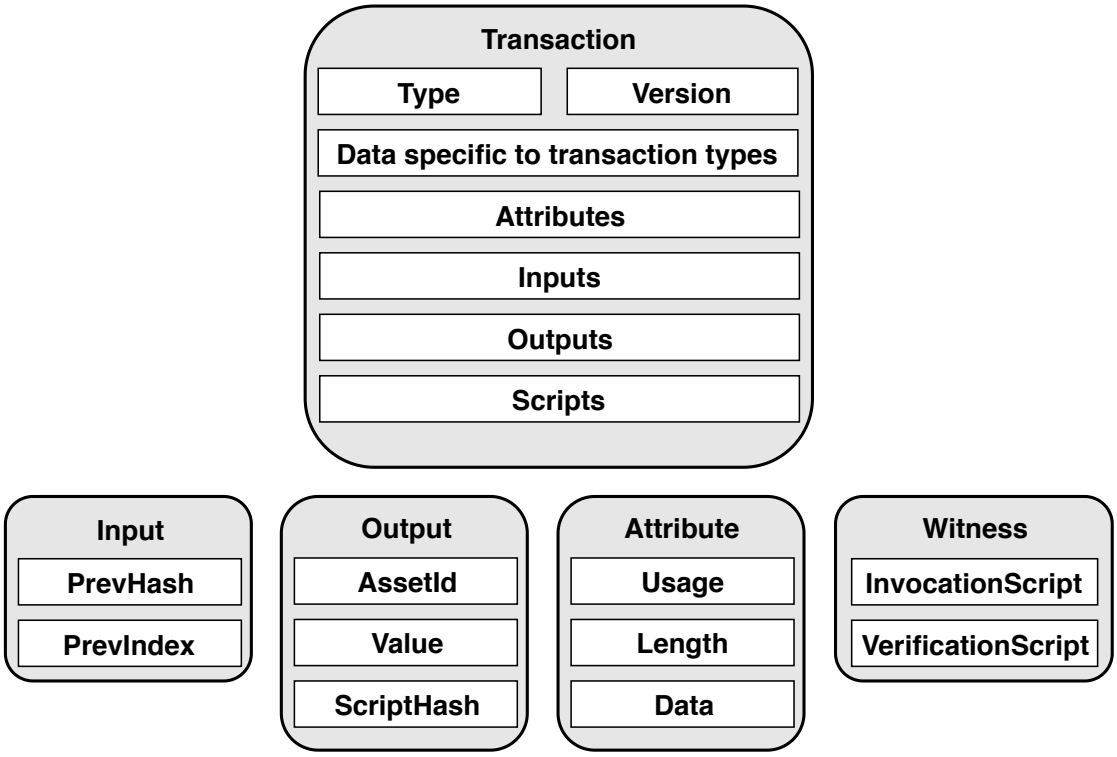

Figure 4.9: The structure of a transaction in the Neo blockchain

the consensus mechanism, are always determined beforehand. These nodes are locked through the NextConsensus field, which includes the hash of a multi signature contract of all nodes participating in the next consensus round. The witness field of the block includes the scripts needed to verify a block. The body of the block contains only a list of transactions. Among these transactions is a miner transaction, which contains the fees as a reward for the speaker. The size of the blocks are restricted by allowing a maximum of 500 transactions per block.

Neo differentiates between nine different transaction types and each of these transaction types requires different fees. Of these transaction types, we have already mentioned the miner transaction. Another transaction type would be an issue transaction to issue an asset, for example the transfer of Neo tokens. In order not to go into too much detail, we will only examine the general structure of a transaction (see Figure 4.9). It can be stated that each of these transaction types fulfills a different task and therefore requires different data as well. Every transaction has a version number and a type, as well as inputs, outputs, attributes and scripts. Neo uses the UTXO model for global assets like NEO and GAS and the account model for assets created by contracts. Each transaction input refers to the UTXO of a previous transaction. An output contains the identifier for an asset, the value and the hash of the public key which was used for the signature. The attributes define additional functions of the transaction whose use is indicated by

the usage field of the attribute. The witness field contains information to verify the transaction. Neo also uses ECDSA signatures, which are verified through a witness. The witness is actually a smart contract and consists of the invocation script and the verification script. The invocation script provides the parameters for the verification script by pushing the values onto the stack. The verification script then verifies the transaction.

4.3.2 Consensus

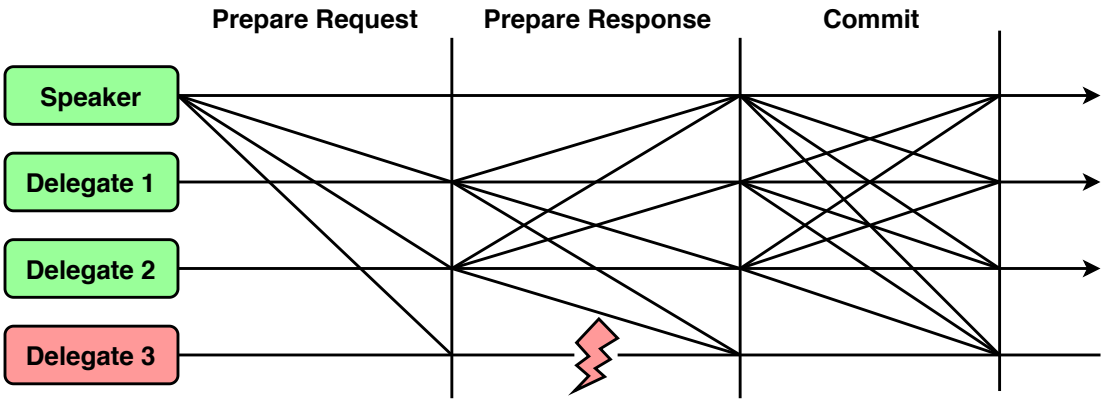

Figure 4.10: Neo consensus mechanism with four consensus nodes and one faulty delegate

The consensus mechanism of Neo is based on the Practical Byzantine Fault Tolerance (pBFT) algorithm proposed by Castro and Liskov [CL+99]. This algorithm describes a replication algorithm which is able to tolerate Byzantine faults and works in an asynchronous environment. The pBFT algorithm requires at least $3f + 1$ replicas to reach consensus, where f denotes the number of faulty nodes. Additional replicas would not bring any additional benefit but would only have a negative effect on the performance, since the algorithm requires the exchange of a lot of messages. The replica set consists of one primary replica and the remaining backup replicas. These replicas move through multiple configurations called views. A client issues a request to the primary replica, which then will proceed to initiate a three-phase protocol. The request will be carried out by all replicas and send the response to the client. In case the client receives $f + 1$ results with the same answer, the replicas have reached consensus. Because it is not always possible to reach consensus, some replicas will request view changes after a certain timeout. If the primary replica receives enough requests to change the view, it will broadcast the request to the other replicas.

Neo implements a modified version of the pBFT algorithm called Delegated Byzantine Fault Tolerance (dBFT) (see Figure 4.10). The dBFT algorithm is more suitable for use in the field of blockchain technology, since the pBFT algorithm suffers from performance and scalability issues. The Neo network categorizes the nodes in ordinary and consensus nodes. Ordinary nodes only have the responsibility to transfer and make transactions,

whereby consensus nodes actively take part in the consensus mechanism. Neo holders vote for the consensus nodes, which take part in the next consensus round. From the set of consensus nodes one speaker (primary replica) is determined, which proposes the next block and forwards the request to the delegates (backup replicas). The delegates then process to verify the received block and broadcast a response with the signature of the block. If a node receives enough responses so that no more then $n - 1/3$ nodes are faulty, it reaches consensus and publishes the new block. If a node can not reach consensus it will issue a request to change the view after a specific time interval. The speaker will restart consensus with a new view, after receiving more than $n - f$ requests, where f denotes the threshold of faulty nodes. It takes the consensus mechanism about 15 to 20 seconds to generate a new block, thus enabling a transaction throughput of 1000 transactions per second.

4.3.3 Smart Contracts

Neo offers the possibility to execute smart contracts on the Neo blockchain with its lightweight virtual machine and smart contract system. The virtual machine of Neo is Turing-complete and is optimized for fast startup times of the execution environment. The smart contracts can be written in different high-level programming languages like C# or Java. The Neo compiler translates the smart contracts to byte code instructions for the NeoVM. The virtual machine is a stack-based engine, which uses a separate interoperation service layer. This layer is responsible to access data external to the virtual machine. Smart contracts can get access to full data on the blockchain as well as other system functions and can use a private storage area, which can only be used by the contract itself. The deployment and execution of smart contracts require a fee to compensate for the used computing power. The fees are payed in GAS and depend on the used resources. In the case of a failure the already used GAS will not be returned. Neo also offers free execution of smart contracts as long as the fees would be lower than 10 GAS. Smart contracts can also call other smart contracts but recursions are not allowed and the relationship must be specified statically, thus a smart contract cannot decide dynamically which other smart contract should be called. This decision is based on the consequences that the behaviour can be determined before the execution and enables parallel execution of smart contracts.

4.4 Hyperledger Fabric

Hyperledger Fabric is a distributed operating system for permissioned blockchains and is one of the Hyperledger projects run by the Linux foundation. It is a highly modular and extensible system, which enables to create blockchain systems tailored for specific use cases and trust models. This is possible through modular consensus models in contrast to hard-coded consensus models of other blockchain systems. Fabric uses a permissioned model, thus every participant in the network can be identified. Furthermore, smart contracts also called chaincode, can be executed on the blockchain. By following an

execute-order-validate architecture, Fabric tackles the limitations of the mainly used order-execute architecture by current blockchain systems. The order-execute architecture can not handle non-deterministic code like we have already seen with Ethereum and Neo, where each node on the network has to execute the same transaction. Another problem is the sequential execution of transactions as it limits the throughput of the system. Fabric has already shown to provide a throughput of 3500 transactions per second. In these systems, confidentiality of execution is also not given, since every node of the network needs to run the code of the smart contract [ABB+18].

4.4.1 Ledger and Blockchain

The ledger in Fabric consists of two components. The blockchain which acts as the transaction log and contains all actions that were performed that led to the current world state and the world state. The world state is stored in a versioned key value store, which is managed by the peer transaction manager. Chaincode is able to store new entries in the format (key, value, ver), where the version number includes the block sequence number as well as the transaction sequence number. Create, update and delete operations on the world state are allowed, which change the world state frequently. The separation of world state and blockchain enables faster state operations, because it does not require traversing the blockchain. The blockchain is stored as append-only files, whereby the world state is stored in a database. Fabric uses LevelDB and Apache CouchDB to realize the key-value store. Hyperledger Fabric offers the concept of channels, where each channel has its own ledger [ABB+18, fab19].

A block consists of the block header, the block data including all transaction and finally the metadata of the block (see Figure 4.11). The block header contains the information which cryptographically secure the blockchain and form the chain structure. It contains the current block number with the current block hash and the link to the previous block through the previous block hash. The block data includes all transactions in a specific order determined by the ordering service, which is responsible for the creation of new blocks. The block metadata includes a certificate, public key and signature of the blockwriter and later on also the indication for each transaction, if it is valid or invalid. A transaction records all operations which changed the world state. The header of the transaction provides metadata about the invoked chaincode. Further, it includes the digital signature of the client, which issued the transaction. A proposal field contains the input parameters for the chain code, which should be invoked by executing the transaction. The response field contains the dependencies and changes to the world state, when the transaction is executed. The state is only changed during the validation phase. The response contains the changes which have to be applied to the state after the validation. Finally, endorsements have to be included, which are signatures from the participants listed in the endorsement policy [fab19].

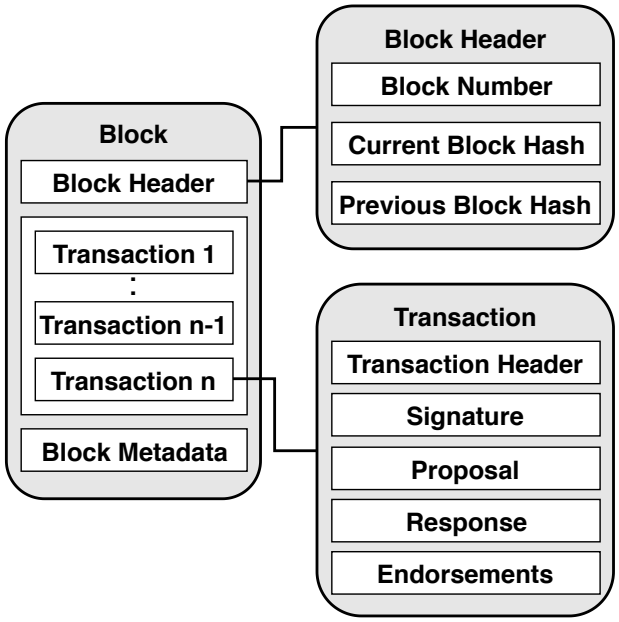

Figure 4.11: Block and transaction structure in Hyperledger Fabric

4.4.2 Identity Management

The permission based model requires a component which is responsible for the management of identities and issuing credentials for authentication and authorization. This task is fulfilled by the Membership Service Provider (MSP), which is composed of a single component from each node. Fabric allows different implementations of the MSP enabling different identity and authentication models. The permissioned model allows an easy identification of potential attackers, which try to carry out Denial of Service (DoS) attacks, through flooding the network with transactions. It also allows the usage of BFT-based consensus mechanisms. In the current implementation of Hyperledger Fabric, the MSP is based on standard Public Key Infrastructure (PKI) methods and digital signatures. Fabric allows the usage of standard X.509 certificates issued by a Certificate Authority (CA) [ABB+18].

4.4.3 Smart Contracts and Chaincode

A distributed application on Fabric consists of the chaincode, which will get executed in the execution phase, and the endorsement policy used for the validation phase. Endorsement policies enable different trust models for different applications as they can determine the endorsers, which are necessary for a specific transaction. It specifies which nodes have to sign the transaction in order to be valid. The smart contracts can be written in different high-level programming languages and get executed in an isolated environment. Currently, Fabric supports the programming languages Go, Node and Java and uses Docker as its isolated execution environment for chaincode. Chaincode has access to a

blockchain and the world state of the system. The state from each chaincode is only visible for the chaincode that created the state. The state can be manipulated or read with GetState, PutState, and DelState operations and chaincode can also access other chaincode and information on different channels, if it has the right permissions to perform these actions. Another important property of chaincode and its execution is that it does not rely on a cryptocurrency [ABB+18]. Chaincode can also define low-level program code, which is not specific to any domain. Fabric has different types of system chaincode for necessary system interactions, for example validation of transactions, access to the ledger and configuration of different channels. Even here, Fabric provides customization options for developers, but needs great care as it affects the systems functionality [fab19].

4.4.4 Transaction Flow

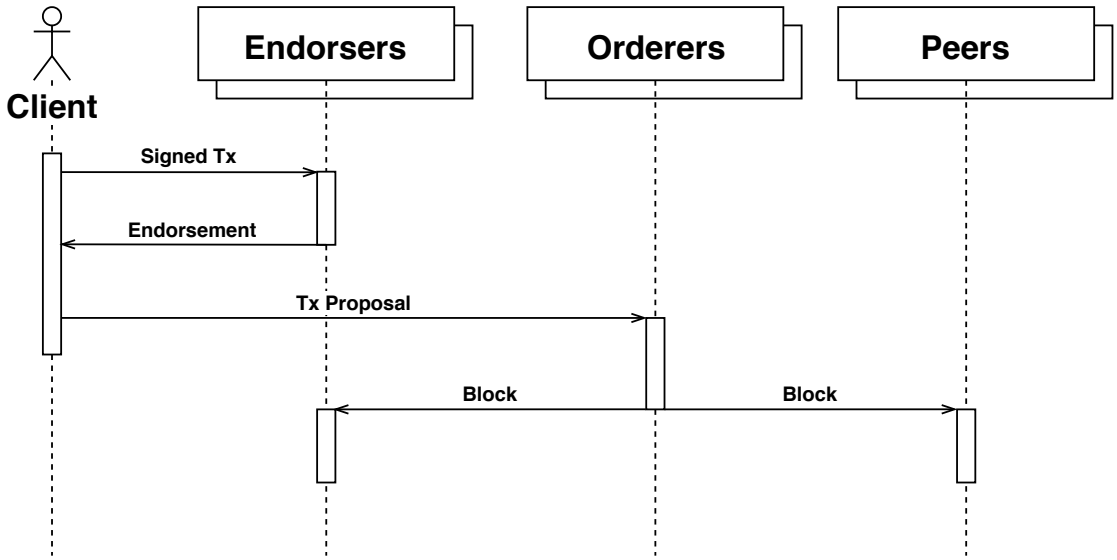

Figure 4.12: Transaction flow of Hyperledger Fabric [ABB+18]

The transaction flow (see Figure 4.12) begins with the execution phase where a client sends the transaction to the peers specified in the endorsement policy. These peers then will simulate the execution of the transaction based on the state of their local blockchain, without storing any state changes. After simulating the execution of the transaction the endorsers create an endorsement, which includes two sets that record the key-value pairs that are read from and written to during the execution of this transaction. The client will collect those endorsements until the endorsement policy is adhered to. After the client has collected enough endorsements, it will send the transaction to a ordering service. The ordering service is not responsible for either running or validating any transactions. Its only purpose is to establish a total order of transactions using a consensus mechanism. Since Fabric has been designed with the intend to provide pluggable components, various implementations of ordering services can be used. In [SBV18], the authors provide

a BFT ordering service, which can achieve up to 10k transactions per second. The transactions are grouped into blocks which later form the blockchain. After that, the ordering service takes care of the dissemination of the blocks to the peers. The peers then enter the validation phase, where the first step is to validate the endorsements based on the endorsement policy. The next step is to check if the two sets describing the dependencies of the transaction, got changed in the mean time. If any of those two checks fail, the transaction is marked as invalid. The last step is to update the ledger. Every transaction is stored in the ledger, even if the transaction was invalid [ABB+18].

4.5 Comparison

Table 4.1: Overview of the differences between the individual platforms

	Bitcoin	**Ethereum**	**Neo**	**Hyperledger Fabric**
Smart Contracts	No	Yes	Yes	Yes
Programming languages	Script	Solidity, Vyper	C#, VB.Net, F#, Java, and Kotlin	Go, Java, and Node
Structure	Blockchain	Blockchain	Blockchain	Multiple Blockchains
Openness	Permissionless	Permissionless	Permissionless	Permissioned
Consensus Mechanism	Nakamoto Consensus	GHOST	BFT	Any
Membership Selection	PoW	PoW / PoS	PoS	Predefined
Scalability	7 tx/s, improves with Lightning network	15 tx/s	10000 tx/s	3500 tx/s
Energy consumption	High	High	Low	Low
Support and Tools	Good	Good	Mediocre	Mediocre

For the comparison of the platforms, nine different characteristics are discussed. These characteristics include smart contracts, programming languages, structure, openness, consensus mechanism, membership selection, scalability, energy consumption and finally support and tools. Furthermore, a short overview (see Table 4.1) of the differences was also created.

Smart Contracts The most important question which we have to deal with is, if the platform is actually in a position to support smart contracts. All platforms we have encountered so far are able to offer smart contracts except for one. It is not possible to create smart contracts for Bitcoin, since it was never intended to offer the execution of code on the blockchain, besides scripts, which offer the possibility for different payment schemes. Ethereum, on the other hand, has already been developed with the ulterior motive to create a smart contract and decentralized application platform. The same thing goes with Neo whose intention is to enable smart economy, where smart contracts form an important component. Hyperledger Fabric goes so far as to call itself an distributed operating system for permissioned blockchains [ABB+18], where smart contracts respectively chaincode can be used to create decentralized applications for the platform. One major distinction is the dependency on a cryptocurrency for the execution of smart contracts. Ethereum and Neo both use the concept of execution fees, paid with the underlying cryptocurrency. Coincidentally, both are called Gas, which in the

case of Neo is a token, while on Ethereum Gas is a special unit to measure transaction fees. On both platforms, this helps to prevent wastage of computing power and provides additional incentives for the miners, since they get rewarded for sharing their computing power. Hyperledger Fabric on the other hand is a general-purpose blockchain platform, without the need of an underlying cryptocurrency. Nevertheless, it is possible to create a cryptocurrency with chaincode that runs on the blockchain. Smart contracts on Ethereum, Neo and Fabric all have their own storage space in the form of a key-value store and can also access data on the blockchain. Intercommunication between smart contracts is able on all three of these platforms, whereby Neo does not support recursive calls between smart contracts and contracts on Fabric must be explicitly allowed to access other smart contracts. Another restrictive property of Neo contracts is that they can not dynamically call other smart contracts, since the behaviour of contracts should be predictable before execution. On the Ethereum platform, it is possible to dynamically call other smart contracts. Smart contracts on Ethereum are visible for everyone and every node has to execute the code. Hyperledger Fabric offers confidentiality, through endorsement policies which define what nodes will execute the code.

Programming languages Every smart contract platform described so far has support for different programming languages. Bitcoin uses its own scripting language Script, which is too limited to actually create smart contracts, because it is not Turing-complete and provides no loops. Ethereum on the other hand offers the possibility to write smart contracts in programming languages specifically designed to work with the EVM. Solidity is a contract-oriented high-level language, which was influenced by C++, Python and JavaScript. It currently has the most support for the Ethereum platform, nevertheless new languages emerge continuously. One of the newcomers is Vyper, which is a contract-oriented, pythonic programming language for the EVM. The intention behind Vyper is to provide a language which makes it easy to build secure decentralized applications that are highly human-readable and very simple in their nature. As we can see, Ethereum differentiates itself from Neo and Fabric by the usage of contract-oriented languages specifically designed to work with the EVM. Neo and Hyperledger Fabric on the other hand offer support for different already existing high-level programming languages. Both offer at least three well-known programming languages, which does not force developers to learn a new language and get used to it. Over time, more and more different programming languages will probably be supported to make the selection even bigger, thus enabling programmers from different domains to create decentralized applications on the platform.

Structure Bitcoin, Ethereum, Neo and Hyperledger Fabric all make use of a blockchain. Fabric provides a significant difference, by allowing the usage of multiple blockchains respectively channels. The basic structure of the blockchain is the same for all of these platforms. Ethereum is different from the others as it supports the inclusion of stale blocks in the blockchain, through the implementation of a modified GHOST protocol. Blocks not only have a parent block but also uncles. Bitcoin uses the UTXO model, whereby Ethereum uses an account-based model and Neo makes use of both models.

Ethereum, Neo and Fabric use key-value stores to store the world state of the system. The state of Bitcoin is only described by the UTXOs. Since we only dealt with the blockchain structure, it is worth mentioning that there are different other structures in use by platforms which have not been described. These include, for example, directed acyclic graphs used by IOTA [Pop17] or a block-lattice structure, which is used by Nano, formerly known as Raiblocks [LeM18]. Since all of the described platforms make use of the blockchain structure, every one of them has to deal with the advantages and disadvantages of this structure, which is particularly noticeable in terms of scalability.

Openness Except for Fabric, which is a distributed operating system explicitly for permissioned blockchains, all other platforms mentioned in this thesis are permissionless blockchain systems. This means that on these platforms everyone has the opportunity to join the network by running a node. Hyperledger Fabric on the other hand requires that every participating node has an identity and the network has to be configured to include the node in the network. Permissioned systems provide more control over the structure of the network and its participants. Confidentiality is also a major aspect, which is more complicated to realize on permissionless blockchain systems, since every node has to verify transactions and execute code. One will have to weigh whether the systems need to be accessible for everyone or just for a specific audience. For businesses which want to cooperate with each other in a trustless environment, a permissioned blockchain would be more suitable than a public system. Public systems tend to be more decentralized, but with the ability to join the network for everyone, they also suffer from scalability and performance problems, as we will see later on.

Consensus Mechanism The consensus mechanism of Bitcoin follows the longest chain rule, where each participant of the network will try to append blocks on the longest fork of the blockchain. Unlike Bitcoin, Ethereum implemented the modified GHOST protocol, which does not select to build on top of the longest chain, but uses a rule that also includes stale blocks in the blockchain, which will also be considered when selecting the branch to build on. The GHOST protocol helps to improve security since Ethereum has a very low block confirmation time of 15 seconds. These approaches will take quite some time to reach block finality, as other branches of the blockchain could overtake the current longest or heaviest fork of the blockchain. If more blocks get added to the branch, the likelihood will become so low over time that a transaction can be considered confirmed. Neo goes a completely different way by using the BFT algorithm and only allowing selected nodes to participate in the consensus mechanism. Each consensus node will vote, if the block should be included in the blockchain. In contrast to Bitcoin and Ethereum, Neo provides single block finality, which eliminates the waiting times. Hyperledger Fabric offers a completely flexible system whereby the mechanism can be exchanged as required. A BFT ordering service for Fabric can be found in [SBV18], which is just one way how an ordering service can be implemented.

Membership selection Bitcoin and Ethereum follow the same approach regarding membership selection. Every node executing the PoW algorithm takes part in the consensus mechanism. By providing the PoW a miner is allowed to propose a block, which should be included in the blockchain. Ethereum is currently in the phase of switching from the PoW algorithm to the PoS algorithm, where each participant provides a certain amount of the cryptocurrency, which gets locked away, to take part in the consensus mechanism. This will have great implications on energy consumption and security of the system. An attacker trying to carry out a 51% attack would have to burn all the money, which is locked away [But16]. PoW would only waste computing power for the time of the attack. Neo also uses PoS in the form that participants vote on the next consensus nodes, with their amount of Neo. As has already been mentioned earlier, Fabric uses permissioned blockchains, which gives control over the participants which are allowed to take part in the consensus mechanism. On Hyperledger Fabric, the ordering service is composed of multiple predefined ordering nodes, which all take part in the consensus mechanism to form the next block.

Scalability Bitcoin and Ethereum both suffer from severe scalability problems. The transactions that need to be processed are steadily increasing but can not be processed fast enough. Bitcoin only has a throughput of 7 transactions per second, whereby Ethereum has a throughput of 15 transactions per second. Both solutions are highly underperforming compared to payment networks like Visa. The introduction of the Lightning network in Bitcoin should help to improve scalability by introducing micropayment channels. Ethereum tries to handle the scaling problem with a different approach, where the blockchain is split into different shards and each shard has its own history of transactions and state. Different nodes would be responsible for different shards, reducing the overall load on the network and allowing the network to process multiple transactions in parallel [CMVM18]. Compared to Bitcoin and Ethereum, Hyperledger Fabric has been able to achieve much better speeds. By implementing and benchmarking a Bitcoin-like cryptocurrency, Fabric achieved 3 500 transactions per second, which is a significant improvement to Bitcoin and Ethereum [ABB+18]. Neo stands out from the aforementioned platforms in terms of speed and promises to achieve up to 10 000 transactions per second, which is about the same as the average transaction speed of the payment network Visa.

Energy consumption The problem with high energy consumption is mainly related to blockchain systems using the PoW algorithm. The PoW algorithm requires to carry out expensive calculations over and over again until a solution is found. If we take a closer look at Bitcoin's energy consumption, we are able to realize that this is an order of magnitude unlike the power consumption of Neo or Hyperledger Fabric. Ethereum has the same problem that comes with using a PoW algorithm, which is also one of the main reasons for Ethereum to transition to a PoS blockchain. It is not easy to calculate the exact power consumption of Bitcoin, but [DV18] estimates a lower bound of 2.55 Gigawatt and a hashrate of 26 quintillion hashes per second performed by the Bitcoin

network. Application-Specific Integrated Circuits (ASICs), like the Antminer S9 provide 14 Terrahashes per second. Besides the operational costs of the mining hardware, also the costs for cooling these systems must be considered. It is clear that platforms that do not use the PoW algorithm consume significantly less power, making Neo and Fabric, and perhaps Ethereum in the near future, more energy-efficient platforms than Bitcoin. Especially in relation to the IoT, energy consumption plays an important role, since the devices are very limited in their computing power and it would therefore be impossible for such devices to participate in the consensus mechanism. However, if we look at other consensus mechanisms that place little demands on the resources of the device, even very limited devices from the IoT could emerge as participants in the network, which could also participate in consensus mechanisms that could certainly help to counter centralization.

Support and Tools When it comes to implementing decentralized applications, developers will also have to consider how many tools are currently existing for the platform and how big is the support of the community. Bad, few or no tools make the development process extremely difficult or at least a lot more complicated. The Bitcoin Ecosystem provides some tools or software[1] related to Bitcoin, but it should be noted that Bitcoin does not provide smart contracts and thus there is no need for such tools. Nevertheless it provides different clients, libraries and development utilities. Ethereum offers an enormous number of different tools compared to the other two smart contract platforms. Describing the whole list here is far too much, so just a few examples should make it clear. Ethereum provides multiple smart contract languages, as has already been mentionend. Further, there exist different frameworks for developing, testing and deploying smart contracts, of which the Truffle framework is the most popular one. Integrated Development Environments (IDEs) like the Remix Web IDE support developers to create decentralized applications in a comfortable way by providing static code analysis or a virtual machine to test the code. One of many blockchain test networks, which can be used is Ganache. A wide selection of APIs for different programming languages enables communication with Ethereum. The list of tools[2] contains a lot more useful tools, which are not mentioned here. Neo can hardly keep up with Ethereum in terms of tools[3] and support. It offers Software Development Kits (SDKs) for different programming languages, but otherwise does not really offer a wide variety of other tools. One advantage, however, is that existing IDEs can be used, as Neo supports different high-level programming languages. Tools and support for Hyperledger Fabric is also limited and can not compete with Ethereum. All of the discussed platforms are open source projects. With a quick look at the insights of the various projects, we can see that Bitcoin and Ethereum are in the race much longer and therefore more popular, established and probably more mature.

[1] https://en.bitcoin.it/wiki/Software
[2] https://github.com/ConsenSys/ethereum-developer-tools-list
[3] https://github.com/CityOfZion/awesome-neo

CHAPTER 5

Requirements Development

In this chapter, we go through the requirements development process to realize a framework for a decentralized IoT data marketplace. We start with a general description of the framework (see Section 5.1). Following, we identify different user classes by looking at already existing solutions and describe these user classes (see Section 5.2). After that, we take a look at the operating environment of the data marketplace and everything that needs to be considered in relation to it (see Section 5.3). Furthermore, we define certain design and implementation rules that must be followed to achieve the desired goals (see Section 5.4). Potential assumptions, dependencies and their implications are also clarified (see Section 5.5). Then follows a discussion of the system features and their functional requirements, which were identified based on existing solutions in the area of traditional and blockchain-based data marketplaces (see Section 5.6). The chapter concludes with the non-functional requirements respectively quality attributes of the system (see Section 5.7). The content of this chapter forms the basis for continuing with the design and implementation of the framework.

5.1 Description

A software framework for a decentralized IoT data marketplace is designed, implemented and evaluated, which gives providers and consumers the possibility to become participants on a decentralized IoT data marketplace in order to sell or buy IoT data products in exchange for money. Using a decentralized approach, blockchain technology is utilized to enable the trading of data without the need of a trusted third party. Blockchain technology enables the automatic enforcement of the rules of the data marketplace by using a decentralized application consisting of multiple smart contracts, which are executed by every node of the peer-to-peer network of a smart contract platform. This allows each participant to be held accountable for his actions by storing every transaction transparently on the blockchain, whereby the blockchain is a tamper-resistant data

structure (see Chapter 2). In order to take on this role, the framework for the decentralized IoT data marketplace must fulfill certain requirements, which are discussed in more detail in the course of this chapter.

Already proposed solutions of data marketplaces still leave room for improvements, as has been discussed in Chapter 3. Therefore, we create a decentralized data marketplace that incorporates the functionalities of existing solutions, but also takes their shortcomings into account. Here, care is taken to ensure that all key elements are provided which are needed to actually use the decentralized IoT data marketplace in the real world. These key elements include user management, device management, product management, negotiation, discovery, selection, routing, settlement, rating, monitoring and web access, which are explained in more detail in Section 5.6. Furthermore, the data marketplace is explicitly created for use in the IoT, which means that special requirements have to be met due to certain characteristics of the IoT.

5.2 User Classes and Characteristics

Table 5.1: Overview of the user classes and their description

User class	**Description**
Provider	A provider is an owner of an IoT device or the IoT device itself. Owners of IoT devices could be individuals, small organizations or large companies, which use the framework to incorporate their devices in the data marketplace, to sell data generated by these devices. The main incentive of the provider to offer its data on the data marketplace is to increase profits by providing data that is otherwise uncollected or unused. By doing so, providers will be able to maximize their revenue.
Consumer	A consumer is an owner of IoT devices or the IoT device itself that is capable of buying and receiving data on the data marketplace. Same as with providers, the owners of these devices could be individuals, small organizations or large companies, which use the framework to purchase specific data. The consumer has the interest to find data, which can be used to improve quality, improve user experience and maximize revenue of own products.
Broker	A broker is the owner of software and hardware infrastructure, which uses the framework to offer his services as a middleman for data trading. The broker is interested in making profit by providing resources to facilitate the transmission of bought data. A broker strives to facilitate as many data trades as possible, with the intention to get a reward for enabling the communication between providers and consumers. Communication between providers and consumers is only possible through the message broker as IoT devices have very limited resources.

The user classes define the different participants of the decentralized data marketplace and their inherent characteristics (see Table 5.1). These user classes have already been mentioned in other works about decentralized data marketplaces [GKJ18, MBC+17], whereby these works follow a different approach regarding the design of the data marketplace. The user classes mentioned in it include providers, consumers and brokers. Providers and consumers are sometimes also called buyers and sellers and identified in nearly every process that involves trading of data, whether in form of concrete devices [HHL18, LDBA17] or abstract entities [RK18].

Furthermore, there is the possibility that users belong to more than one user class. Gupta et al. [GKJ18] mention intermediate data processors, which function as producer and consumer simultaneously by buying data and processing the bought data for resale. Missier et al. [MBC+17] give an example of value-added services, which buy data to offer more sophisticated services for other customers. In conclusion, however, it can be stated that three different user classes can be identified, whereby providers and consumers are always present and, depending on the design, brokers are also needed.

5.3 Operating Environment

Before starting with the design and the implementation of the framework, it is important to determine in which environment the data marketplace will be operated. Otherwise, it might turn out in the end that the design is not suitable or the implementation is not appropriate for the environment and therefore can not be used. To be sure that the individual components of the IoT data marketplace can be operated without problems, great importance should be attached to the compatibility with popular server operating systems and the backing technologies of IoT devices. In Chapter 4, we have already compared several smart contract and decentralized application platforms, whereby we shall focus on a platform that enables reliable operation of the data marketplace and good support for the developers of the system. The following four things regarding the environment have to be considered:

OE-1 The data marketplace shall be operable on at least one popular server operating system.

OE-2 The data marketplace shall consider current technologies used by IoT devices.

OE-3 The data marketplace shall take into account the available resources of IoT devices.

OE-4 The used smart contract and decentralized application platform shall be reliable with good support and tools.

5.4 Design and Implementation Constraints

The aim of this work is to find out how far blockchain technology is suitable for the implementation of a decentralized IoT data marketplace. The characteristics of the IoT, as well as the advantages and disadvantages of traditional and blockchain-based data marketplaces have already been discussed in Chapter 2 and 3 and provide the reasoning behind the exploration of blockchain technology for this purpose. We want to use blockchain technology not only as a payment system, but also to realize more complex functionalities via the execution of smart contracts, as has already been done in other works about blockchain-based data marketplaces [GKJ18, BR18, MBC+17]. For this purpose, the following design and implementation constraints can be stated:

CO-1 The architecture of the IoT data marketplace shall follow a decentralized approach by exploring blockchain technology.

CO-2 The data marketplace shall use smart contracts executed on a smart contract and decentralized application platform to implement data trading.

5.5 Assumptions and Dependencies

The data marketplace shall use a smart contract and decentralized application platform to implement specific functionalities. Since it is not within the scope of this work to design and develop a new smart contract and decentralized application platform, an already existing solution shall be used. As a result, the data marketplace depends on this external system. Further, it must be considered that future changes to the smart contract platform will have a direct impact on the data marketplace.

The data marketplace will depend on the external system in many ways. Security, performance and scalability of the platform will influence the operation of the data marketplace. If, for example, the smart contract platform is under attack or is under heavy load, the operation of the data marketplace will also experience problems. In summary, the following dependency can be recorded:

DE-1 The operation of the data marketplace depends on the used smart contract and decentralized application platform in various ways.

5.6 System Features

The system features describe the main features of the data marketplace and their functional requirements. For the framework, essential system features and their corresponding functional requirements are derived by looking at current solutions for data marketplaces.

5.6.1 User Management

To become part of the data marketplace, the participant needs to be registered. Due to the constraint that a smart contract platform respectively blockchain technology has to be used to implement the data marketplace, each participant of the data marketplace must have an account for the chosen smart contract platform. The marketplace may also need more information about the participant, which also requires a profile like in [GKJ18]. Further, the authors of [MŽ16] have stated that providers and consumers register via a web interface for their implementation of a centralized data marketplace and in [BR18] buyers, sellers and mediators also have to register themselves by using smart contracts. Therefore we can record the following functional requirements:

FR-1 The user shall be able to create a profile on the data marketplace.

FR-2 The user shall be able to update its profile on the data marketplace.

FR-3 The user shall be able to delete its profile from the data marketplace.

FR-4 The user shall be able to find profiles on the data marketplace.

5.6.2 Device Management

Providers must be able to manage their devices that are the actual creators of the data to be traded on the data marketplace. Each provider can have multiple devices through which it generates different data. With this, the provider can then proceed to create various products by using its created devices. The registration of devices is also an important part of the proposed solution in [GKJ18], although there is no distinction made between devices and products. Also in the work of Ramachandran et al. [RRK18] this distinction does not occur. However, this distinction clearly makes sense with the background that devices can offer several different products respectively data streams and not every buyer is interested in the whole batch of information that is generated by one device. Furthermore, consumers also get more detailed information about the device that generates the data. It also allows consumers to search for products generated by a particular device. The following functional requirements can be identified for the management of devices:

FR-5 The user shall be able to create devices on the data marketplace.

FR-6 The user shall be able to update its devices on the data marketplace.

FR-7 The user shall be able to delete its devices from the data marketplace.

FR-8 The user shall be able to find devices on the data marketplace.

5.6.3 Product Management

The product management includes all interactions that enable providers to take care of their products offered on the data marketplace. A provider can create several products, which can be bought by consumers, whereby every product is generated by a specific device. In [GKJ18] and [RRK18], providers can publish devices through the usage of a smart contract, whereby the latter solution also uses a DFS to store metadata about the products on the data marketplace. Updating and deleting products is not clearly mentionend, but is of great importance to enable providers to adapt their products to certain conditions. A provider might want to adjust the interval because the mode of operation of the device has changed or adjust the price to respond to market fluctuations. If the provider is not longer willing to continue to offer the product or the device is no longer operated, it should be possible to delete the product from the data marketplace. The removal of products that are no longer offered reduces unnecessary interactions for potential consumers, which have to deal with outdated information. Finally, we state the following functional requirements:

FR-9 The user shall be able to create products on the data marketplace.

FR-10 The user shall be able to update its products on the data marketplace.

FR-11 The user shall be able to delete its products from the data marketplace.

FR-12 The user shall be able to find products on the data marketplace.

5.6.4 Negotiation

Negotiating on a marketplace is not particularly unusual when looking at the real world. However, if we look at already existing solutions of data marketplaces, we usually notice that most of them only focus on the sale of data at a fixed price. The authors of [BR18] mention that a data marketplace has many more components, which have not been discussed in their work, whereby they also name price negotiation. Further, protocols for streaming data payments as foreseen in the SDPP by Radhakrishnan and Krishnamachari only offer the buyer to select an offer from a menu without negotation [RK18]. The proposal of a decentralized infrastructure for fair and trusted IoT data trading from Missier et al. [MBC+17] also only describes a simple payment model, whereby each individual message has a constant unit value. The opportunity to negotiate with each other makes it possible for the participants to represent their interests and thus to identify better deals between the negotiating parties. For example, a buyer who wants to purchase large amounts of data over a longer period of time might ask the seller for a volume discount. The seller would be tempted to accept the buyer's offer because he would spend

a large amount of money on his product at one go. This negotiates a good deal for both parties, which safeguards the interests of both. Gupta et al. [GKJ18] already describes a data trading concept, whereby the buyer and seller are directly communicating with each other and engage in a negotiation process to agree on an offer. The two parties negotiating with each other exchange bids, which either side can accept or refuse. If both parties have agreed, the exchange of data can begin. In order to realize negotiation, the following functional requirements should be recognized:

FR-13 The user shall be able to make a request to start a negotiation regarding a specific product.

FR-14 The user shall be able to make bids on specific products.

FR-15 The user shall be able to make counterbids on specific products.

FR-16 The user shall be able to accept or refuse an offer.

5.6.5 Routing and Transmission

The routing and transmission of data is an essential part of a data marketplace. The buyer must be able to easily receive its purchased product. Here, particular attention must be paid to the integrity, confidentiality and availability of the data. This is considered in more detail later on in the course of this chapter (see Section 5.7).

In the literature, there are different approaches on how the communication between buyers and sellers can be realized. In [GKJ18], the buyers and sellers establish a direct TCP connection and the entire data exchange takes place via this connection. A complete opposite to this is the decentralized infrastructure for fair and trusted IoT data trading proposed in [MBC+17]. The architecture relies upon the largely adopted IoT-brokered data infrastructure and the entire data exchange takes place via a broker, which forwards the data to the buyer. How to exchange the data is ultimately a design decision (see Section 6.2). It is clear that without a reasonable exchange of data, no trade between contractors can be handled.

FR-17 The data marketplace shall be able to transmit the product from the provider to the consumer.

FR-18 The data marketplace shall be able to route messages from providers to the right consumers.

5.6.6 Settlement

The conclusion of a trade is formed by the settlement. Both parties have to agree on the payment and the quality of the trade. The seller would like to charge the buyer for the transferred data and the buyer must be able to pay the seller. Furthermore, it is important that invoices can be generated. In most countries it is a duty for the seller to issue an invoice to the buyer.

Additionally, the protection of the participants engaging in a data trade is an important issue that needs to be considered. The participants may fail to abide the conditions of the trade, be it intentional or unintentional false behaviour. Gupta et al. [GKJ18] describe a relatively simple approach whereby both consumer and provider use a metering system to count N transactions. The counters are compared and if they match, the system proceeds with the invoice and the payment of the provider. Otherwise, a dispute will be lodged and the payment is refrained and participants will be penalized. A more advanced approach is described in [BR18], where predicates are defined and codified, which have to be satisfied by the data. Missier et al. [MBC+17] check the consistency of data cubes, which record the data flow between the participants to ensure fairness. The interactions that take place in the settlement process are particularly important for the successful completion of the transaction. Without the possibility to come to an agreement, the trade between two parties only makes little sense at the end. Therefore, we consider the following functional requirements:

FR-19 The provider shall be able to receive payments from consumers.

FR-20 The consumer shall be able to make payments to providers.

FR-21 The data marketplace shall enable the generation of invoices.

5.6.7 Rating

A rating system is extremely common in today's data marketplaces. This applies to both traditional and blockchain-based data marketplaces, which have already been discussed in Chapter 3. Gupta et al. [GKJ18] mention a reputation system to penalize misbehaving participants of the data marketplace, which are recognized during the settlement process through the transaction counters provided by the metering system. In [RRK18], the authors propose to store ratings of buyers and sellers on the blockchain, as they can not be tampered with. Even though Missier et al. [MBC+17] do not provide an implementation of a reputation system, they emphasize the possible application of a reputation system to penalize misbehaving parties or encourage honest behaviour. The authors of [MŽ16] mention the use of a simple credibility model, whereby the credibility is calculated as the ratio between completed measurements and measurements the seller agreed to deliver.

A rating system enables participants of the data marketplace to obtain an assessment of the reliability of other participants and the quality of the products, which are offered by them. A better rating results in greater confidence in the participant and the product offered, which increases the likelihood that they will be selected to trade. This will also increase the number of successful trades as unreliable participants and poor quality products will perish. In order to implement a rating system we can record the following functional requirements:

FR-22 The data marketplace shall be able to calculate the rating of every user.

FR-23 The data marketplace shall be able to store the rating of every user.

5.6.8 Monitoring

Monitoring provides a data marketplace with the ability to track certain metrics, which can be used to compare and rate its participants and products. The most frequent metric tracked by current solutions of data marketplaces is the number of data transfers between users. Gupta et al. [GKJ18] use a metering system to count the transactions between providers and consumers, which is subsequently used for the settlement. Therefore, the monitoring ability lays out the basis for the protection of providers and consumers, since it can be determined whether both parties agree on the quality of the trade by comparing the recorded metrics and if this is not the case a dispute can be lodged. Missier et al. [MBC+17] propose a decentralized infrastructure for fair and trusted IoT data trading to realize a data marketplace, which is mainly enabled through monitoring the data flow between all participants, whereby all messages are logged and traffic cubes are constructed. By checking the consistency of traffic cubes, the data marketplace is able to detect misbehaviour. Therefore, we add the following functional requirements:

FR-24 The data marketplace shall be able to track certain metrics for every trade.

FR-25 The data marketplace shall be able to store the recorded metrics for every trade.

5.6.9 Discovery and Selection

Discovery and selection of products is an important part for every consumer in the system. Unfortunately, in the discussion about already existing solutions of data marketplaces (see Chapter 3), we have been able to determine that most neglect this point. However, there have also been works that have deliberately taken this element into account. In [GKJ18], consumers can issue a query to the broker, which then matches the products to the query and returns it to the consumer. It is essential for consumers to find the right products that satisfy their needs and select them for purchase. Further, it is particularly noteworthy that in this proposal the discovery and selection functionality is outsourced to an extra device with a large amount of computational resources, since it is a fairly expensive process. Ramachandran et al. [RRK18] also make an effort to

enable buyers to find relevant products, by providing a metadata storage, which stores additional information about every product offered on the data marketplace. Buyers can use this metadata to efficiently look for products of interest. We can record the following functional requirements:

FR-26 The data marketplace shall store important data about products to enable efficient discovery.

FR-27 The user shall be able to discover products by providing a search query.

FR-28 The user shall be able to select a discovered product for purchase.

5.6.10 Web Access

The participants of a data marketplace shall be able to interact with the system through user-friendly interfaces. In [MŽ16], the authors provide access to the data marketplace through the usage of WS-* or Representational State Transfer (REST) web services, which enable the registration of devices and consumers. Ramachandran et al. [RRK18] do not provide a user interface in their implementation of a decentralized data marketplace. Nevertheless, the authors mention that a data marketplace needs a user-friendly interface which makes it easy to use the various functionalities of the data marketplace. Following from this, we define the last functional requirement:

FR-29 The data marketplace shall provide a web interface for its various functionalities.

5.7 Quality attributes

The quality attributes describe the non-functional requirements of the system. The authors of [BR18] mention essential attributes of a data marketplace. Transparency, privacy and security are major aspects, which have to be considered within the design. A transparent data marketplace prevents unlawful exchange of data, by providing a public log which records the transmission of ownership of data traded through the system. Additional details of the data trade are captured and used later on, to make the participants accountable for their actions. Furthermore, secure transmissions improve privacy and security of the system, since third parties that are not involved in the specific data exchange, are not able to access the data. Gupta et al. [GKJ18] also mention the use of symmetric encryption to ensure the integrity of the data. The authors also mention reliability as a key component of data marketplaces, whereby they name broken connections as one potential problem. The performance of the system must be taken into account so that the solution can also be used in a realistic scenario. Among other things, attention must be paid to how the data is transmitted. If the data exchange takes place via a blockchain, the transmission speed is bound through the transaction speed. For example, Zhao et al. [ZLM+18] use a blockchain for the SPS architecture, whereby

publishers send events to the blockchain and subscribers receive notifications from the blockchain. Furthermore, many works also deal with fairness respectively credibility of the system, no matter which approach is considered [GKJ18, BR18, MŽ16, MBC+17]. Fairness is ensured through the use of a rating system, tracking of all data transfers or programmable logic. Consequently, the following quality characteristics can be recorded, which must be considered in the design:

Performance

PER-1 The performance of data transmissions shall not be restricted through the smart contract platform.

Security

SEC-1 The data marketplace shall encrypt the data which is transmitted between providers and consumers.

SEC-2 The data marketplace shall use an immutable public log to ensure transparency and integrity.

SEC-3 The data marketplace shall ensure fairness between providers, consumers and brokers.

Robustness

ROB-1 The data marketplace shall be able to recover, if the connection between providers, consumers or brokers fail.

CHAPTER 6

Design of the Framework

After defining all the requirements for a decentralized IoT data marketplace, we proceed with the design of the framework. We give an explanation of the architecture in which we provide an abstract overview of the entire framework (see Section 6.1). In the course of the architecture description, we discuss the individual components including proxies, brokers, IoT devices, smart contracts and the GUI, whereby the discussion only includes a description of the components and their characteristics and responsibilities. In the course of the next chapter, however, we will take a closer look at the implementation of the components (see Chapter 7).

Since the architecture description does not mention how the framework satisfies the requirements of a decentralized IoT data marketplace, we follow up with a detailed explanation of the system design (see Section 6.2). For this purpose, we elucidate how the components which have been described earlier, are able to provide the key elements of a decentralized IoT data marketplace. We look at the actions taken by each component and how they interact with each other.

6.1 The Architecture

6.1.1 Overview

The design of the framework is based on a three-tier architecture (see Figure 6.1) whereby each tier describes the functionalities of its components. The components are independent from each other and operate in a different environment and on different platforms. This contributes to better scalability, fault tolerance, flexibility, interchangeability and a loose coupling of the components, which ensures good interchangeability. All tiers are assigned to a specific user group (see Section 5.2) which have different interests in the system and requirements for a decentralized IoT data marketplace.

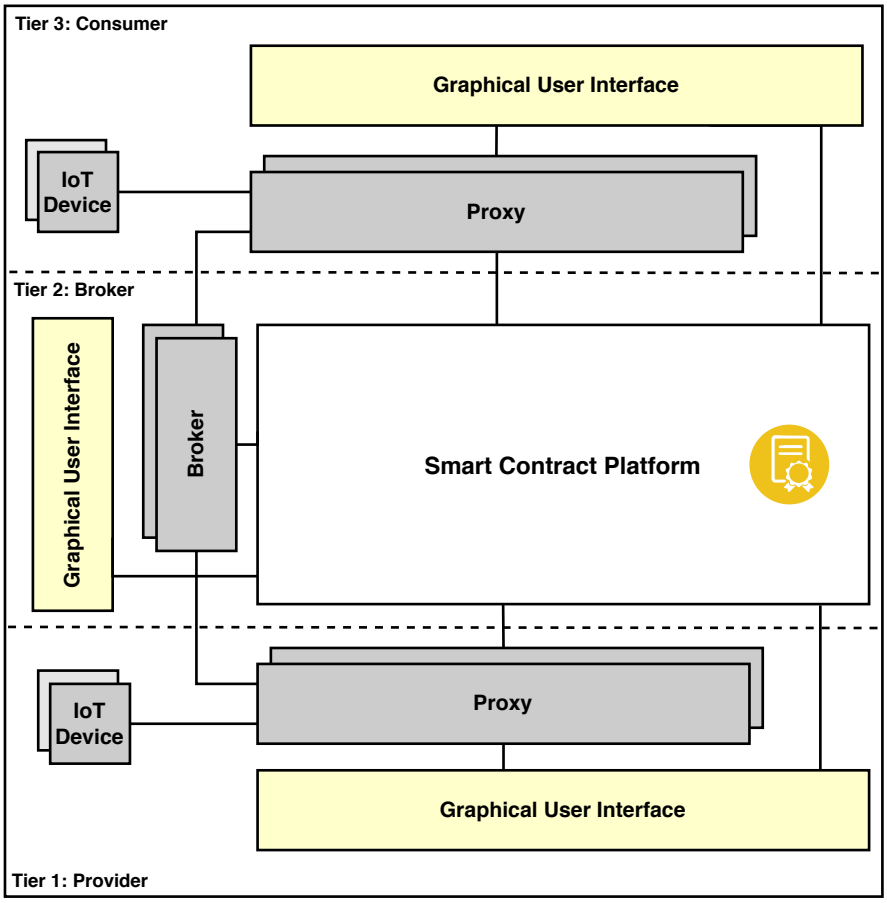

Figure 6.1: Abstract architecture for a three-tier framework for a decentralized IoT data marketplace

The first tier defines the components for providers. It consists of the IoT devices from the different providers which are generating the data that should ultimately be sold and purchased. Further, the first tier includes the proxies which act as the link between the first tier and the second tier and enable the communication between IoT devices and the smart contract platform and the brokers. Furthermore, a GUI allows the provider to interact with the proxy and the smart contract platform.

The operators of brokers and their corresponding components are represented in the second tier of the framework. Further, the second tier includes the smart contract platform with its peer-to-peer network. The second tier facilitates the data trading process between providers and consumers by providing the infrastructure and decentralized applications for the decentralized IoT data marketplace.

The third tier includes the components for consumers. This layer forms the counterpart to the first layer, whereby IoT devices from consumers want to purchase data. It also includes the proxies acting as the link between the second tier and the third tier. Like the provider, the consumer uses a GUI to interact with the proxies and the smart contract platform.

6.1.2 Proxy

The proxy is an important component which enables providers and consumers the integration of IoT devices that have very little computational resources in the decentralized IoT data marketplace. A provider respectively consumer can operate one or more proxies, depending on how far its IoT devices are distributed. This ensures that IoT devices that are geographically not close enough to each other do not need to use the same proxy which would result in inefficient communication.

The most important task of the proxy is to represent the underlying IoT devices in all matters viable, so they have to fulfill as few resource-intensive tasks as possible. The reasoning behind this is that IoT devices usually have very limited computational resources, which makes it very difficult to implement various functionalities necessary for participation in the data marketplace. IoT devices would have to perform cryptographic operations to interact with the smart contract platform, since every transaction needs to be signed. Furthermore, IoT devices would have to carry out a high number of cryptographic operations since the communication between providers and consumers needs to be encrypted to ensure privacy, confidentiality and integrity of the data. This is particularly problematic if the IoT devices need to maintain multiple encrypted connections simultaneously, whereby the multiple connections alone are already stressful enough. If the IoT devices have enough resources, it would also be possible to communicate directly with the brokers and the smart contract platform.

The proxy also brings benefits in terms of security. Individuals, small organizations or large companies are more reluctant to allow direct access to their devices, especially for security reasons. Here comes the further advantage of a proxy into play, which increases security by hiding the infrastructure behind the proxy. This means that it is not possible to access the IoT devices directly from outside the network.

6.1.3 Broker

The broker is the component which facilitates the data trading process and is responsible for the highly resource-intensive tasks which can not be taken over by proxies or IoT devices. This means that the broker must be operated on a device with enough computational resources. Since more computing power also means higher costs, an incentive mechanism is used that rewards brokers for providing their resources. Therefore, not only the provider is paid for the sale of its data but also the broker who has made its computing power available, so that the trade became possible in the first place. This gives owners of hardware infrastructures an incentive to run a broker on expensive hardware that might otherwise be left unused.

The proxy takes over a lot of resource-intensive tasks from IoT devices, especially regarding cryptography. Since a proxy is operated directly by the provider or consumer, it is also advantageous to relocate the heaviest and most resource-intensive tasks to a broker including the management of numerous concurrent connections and product discovery. The more participants the data marketplace has, the more connections have to be managed by IoT devices or the proxy, whereby the number of connections can become very high. The broker acts as a middleman between providers and consumers, and forwards the messages to the right consumer. The broker also prevents message loss if a consumer is offline or not able to receive the messages.

Also, the discovery of products which are listed on the decentralized IoT data marketplace is particularly complex, as the number of them can become very large. It is especially important for consumers that they are able to find products that meet their requirements. Checking these criteria must be done for each product listed on the data marketplace, which makes it a very resource intensive task and is therefore also taken over by the broker.

Monitoring the data transfer is also an important responsibility of a broker. This does not differ from the monitoring performed by providers and consumers. However, the metrics recorded by a broker allow us to determine if the provider or consumer is behaving properly, as the broker knows both what data the provider has sent and which data the consumer has received. It works only as long as at least two parties behave honestly. This approach is not enough to ensure fairness, if a provider or a consumer cooperates with a broker to cheat the other party.

6.1.4 IoT Device

The IoT device is a context-aware device which has the ability to store and process data, is able to communicate with other entities over standardized protocols and is uniquely addressable. However, these devices mostly do not have a lot of computational resources and storage capacity which has to be considered in the design of a decentralized IoT data marketplace. IoT devices can be smart devices, sensors or similar objects, which among other things are able to record various physical or chemical properties about their surroundings. The IoT devices use a proxy to use the functionalities of the decentralized

IoT data marketplace and communicate through the proxy to invoke functions of smart contracts or transmit or receive data from a broker in order to buy or sell products on the data marketplace.

6.1.5 Graphical User Interface

The users of the decentralized IoT data marketplace are able to interact with the system through a user-friendly interface. A GUI furnishes the possibility to use the various functionalities provided by the data marketplace. Through the GUI, users are able to register themselves in order to be able to actively participate in the system. Furthermore, the GUI offers users the possibility to browse the data marketplace and view the different users, devices and products. Further, the user can interact with the GUI to manage its own profile and devices.

6.1.6 Smart Contracts

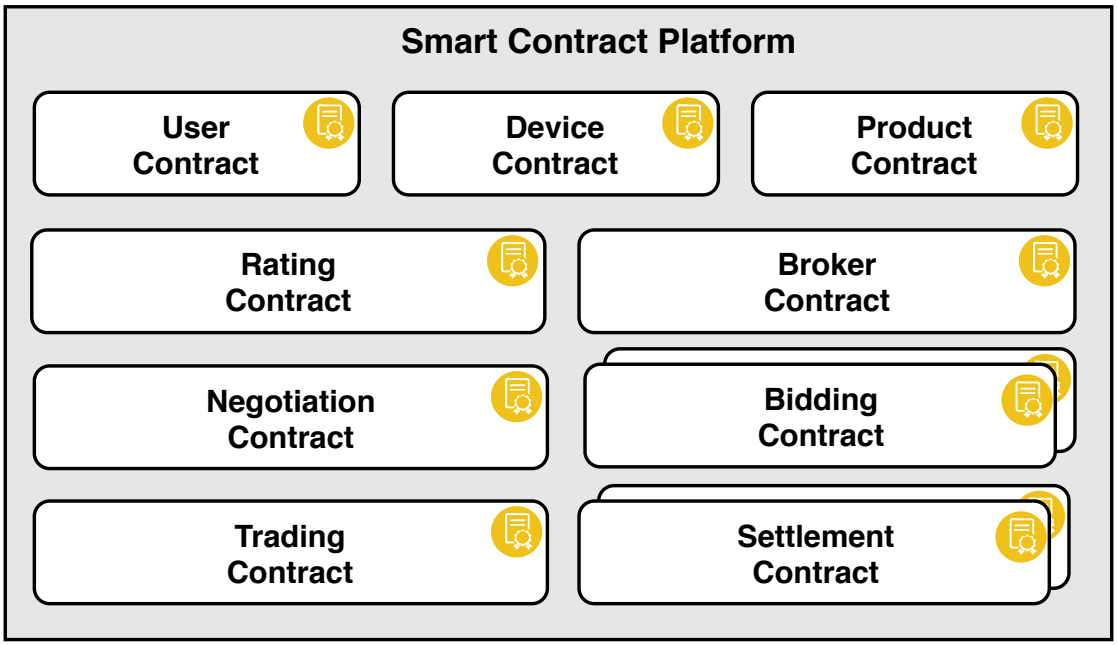

Figure 6.2: Smart contracts for a decentralized IoT data marketplace

The decentralized IoT data marketplace uses different smart contracts deployed on a smart contract and decentralized application platform (see Figure 6.2) in order to realize specific functionalities and enforce the rules of the data marketplace.

In order to participate in the system, users have to register themselves and provide personal information for their user profile. The user profile enables the possibility to get contact details of other users and the necessary information to generate invoices. The user contract is responsible for managing the user profiles storage and keeps the

personal information about individuals, small organizations and large companies on the data marketplace. The smart contract provides the means to create new user profiles and it gives users the ability to update their profile in order to keep their personal information up-to-date. Additionally, this contract makes it possible to retrieve the profile of specific users. If a user no longer wishes to use the system, existing profiles can also be deleted. Since data from a blockchain cannot really be deleted, we will take a closer look at this process later (see Section 7.1).

Another smart contract is responsible for the device management. Users need to be able to manage their devices that generate the different data products. The information provided about the devices give other users the possibility to get more detailed information about the origin of the product and the reliability of the device. Through the usage of the smart contract a user is able to add new devices. Furthermore, the smart contract allows a user to update and delete devices. It is important that devices can be found on the data marketplace, which is also accomplished through the usage of the device contract.

As with users and devices, another smart contract is needed to manage the data products offered by providers. The product contract manages all products of the data marketplace and gives more information about the products being offered for sale. Like the smart contracts which have already been discussed before it also offers the possibility to carry out create, read, update and delete operations.

Further, a broker contract provides the possibility to register instances of brokers on the decentralized IoT data marketplace and gives further information about them. It is necessary to communicate with a broker to use its functionalities which is only possible if the broker can be found by providers and consumers. Therefore the broker registry enables the discovery of different brokers based on certain criteria, such as their location.

Another smart contract is used to realize a rating system in order to rate devices and brokers. The contract is responsible for the calculation and storage of the rating. The rating system provides participants with the possibility to obtain an assessment of the reliability of other participants.

The data trading process is also realized through the usage of smart contracts. The negotiation contract provides the participants with the possibility to start negotiations with each other. It is responsible for managing the bidding contracts whereby a bidding contract enables providers and consumers to exchange bids and creates a transparent view on the negotiation process. Both parties are able to see how an agreement or no agreement was reached. Furthermore, a trading contract is responsible for managing all trades. Each trade is assigned a settlement contract, which acts as an conditional escrow and facilitates the settlement process. The implemented smart contracts are introduced in Section 7.1.

6.2 System Design

6.2.1 Creating a User

In order to use the decentralized IoT data marketplace, a user must be created. The user profile is a precondition which has to be met to incorporate IoT devices in the data marketplace.

Managing the user information requires only very few interactions with the user contract. The GUI provides the user with the possibility to send transactions to the user contract in order to invoke the create, find, update and delete operations offered by the smart contract. For this purpose, the user must have an account on the smart contract and decentralized application platform which can be used by the GUI to interact with the smart contract. Further, the user contract ensures that users can only manage their own profile which is associated with their private key. How exactly the smart contracts store the data will be discussed in Chapter 7.

6.2.2 Integrating IoT Devices

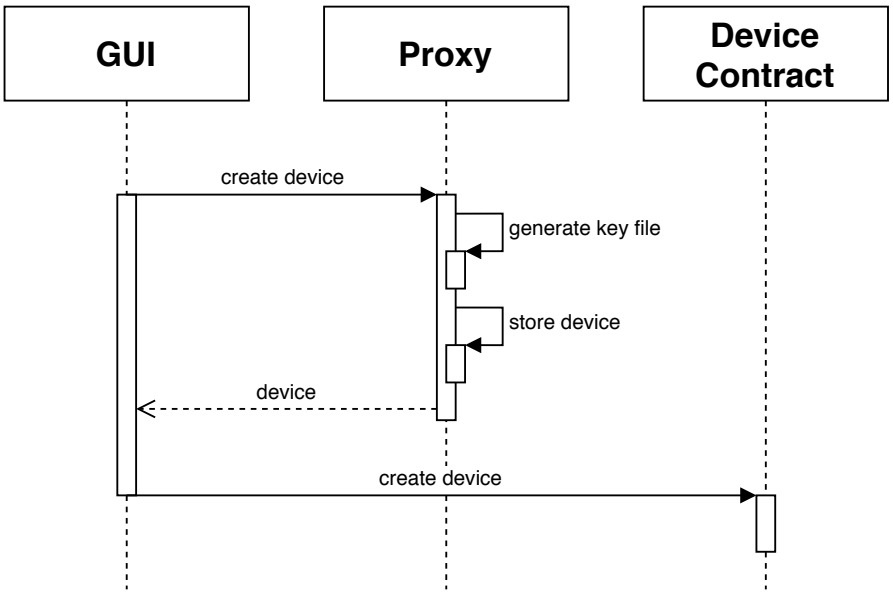

Figure 6.3: Sequence of interactions to integrate an IoT device

Once a user has created a profile, it is possible to use all functionalities of the device contract. This smart contract offers the user the possibility to add different devices to the data marketplace and to manage them afterwards. A distinction can be made between two different types of devices: consumers who want to buy data on the data marketplace and providers who want to offer data products for sale. The device contract offers the functionalities to create, read, update and delete devices. For this, too, only a transaction has to be sent to the smart contract, as is made possible by the GUI.

Since the added devices usually have very low computational resources, they cannot independently create a transaction. Therefore, a device sends requests through a proxy which creates a transaction for the corresponding request and forwards it to the smart contract platform. The point that must be considered here is that the devices must also be registered on the proxy (see Figure 6.3), since the proxy must be able to authenticate the various devices in order to send transactions to the smart contract platform on their behalf. This means that the proxy manages a separate key pair for each device, which can only be used if the device is successfully authenticated. A device uses the credentials which are created during the registration process to authenticate itself. Should a device be able to carry out the necessary operations itself and has direct access to the smart contract platform, the detour via the proxy does not have to take place. Of course, this applies to all operations that IoT devices may want to carry out on the smart contract platform.

6.2.3 Managing Products

Each user which wants to sell data can use the product contract to list new products on the decentralized IoT data marketplace. The created products can also be updated, deleted and searched and the smart contract ensures that a user can only manage its own products. To use these functionalities of the data marketplace, users also only have to send a corresponding transaction to the smart contract platform, which then carries out the respective operation.

6.2.4 Discovering Products

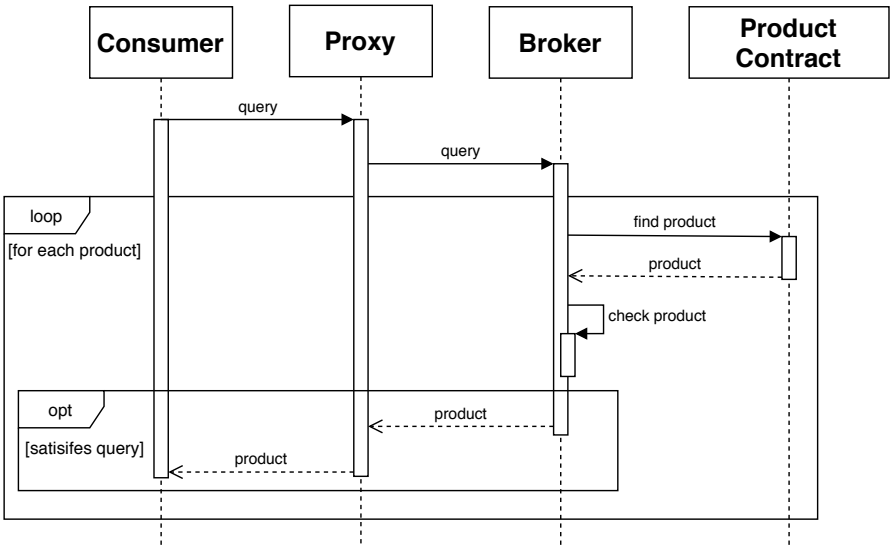

Figure 6.4: The process of searching for a product on the data marketplace

As we have already mentioned in the architecture description (see Section 6.1), the discovery process (see Figure 6.4) is a very resource intensive task facilitated by a broker, since it is operated on a highly resourced device. It allows consumers to search for specific products that meet defined criteria, helping them to find the right product. For this purpose, the consumer sends a search query to the proxy, which is responsible for forwarding this request to a broker. Since several brokers can be considered, the consumer can also use the proxy to search for a suitable broker. The consumer can decide which broker is most suitable based on several criteria, but the most important is the geographical location of the broker to ensure low latency. The search process for a broker is the same as for a product only that it is carried out directly by the proxy.

After the broker has received the search query of a proxy, it uses the product contract to iterate over all products listed on the data marketplace. Here, the broker examines each product according to the desired properties which were specified in the request. After the broker has checked whether the product satisfies the query, it sends the product back in response to the proxy, which forwards the result to the consumer. The consumer can use this result to select a product that meets his previously defined requirements and, if interested, continue to purchase the product.

6.2.5 Negotiation

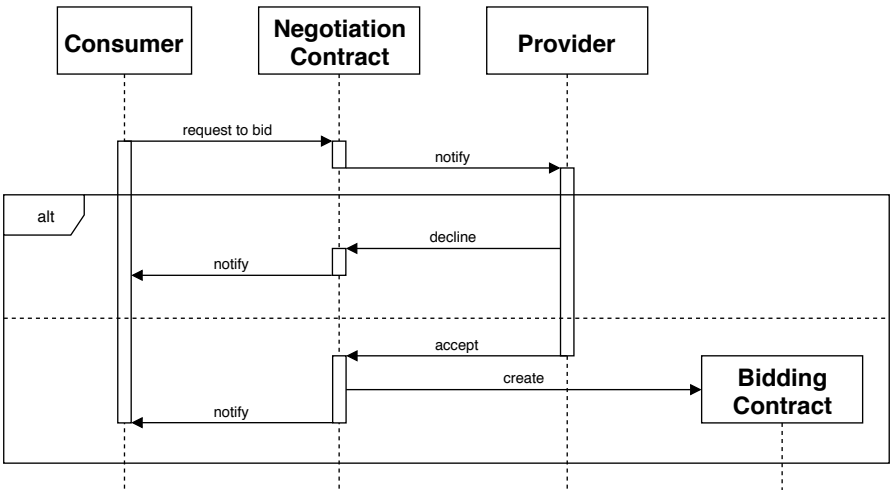

Figure 6.5: The actions taken to start a negotiation process

The negotiation process (see Figure 6.5) is always initiated by the consumer who wishes to buy a particular product. For this purpose, the consumer sends a transaction to the negotiation contract in order to request a negotiation with the provider of the product. The smart contract ensures that only the provider of the product can respond to the request which can either accept or reject the request. If the provider accepts the request, a new bidding contract will be created.

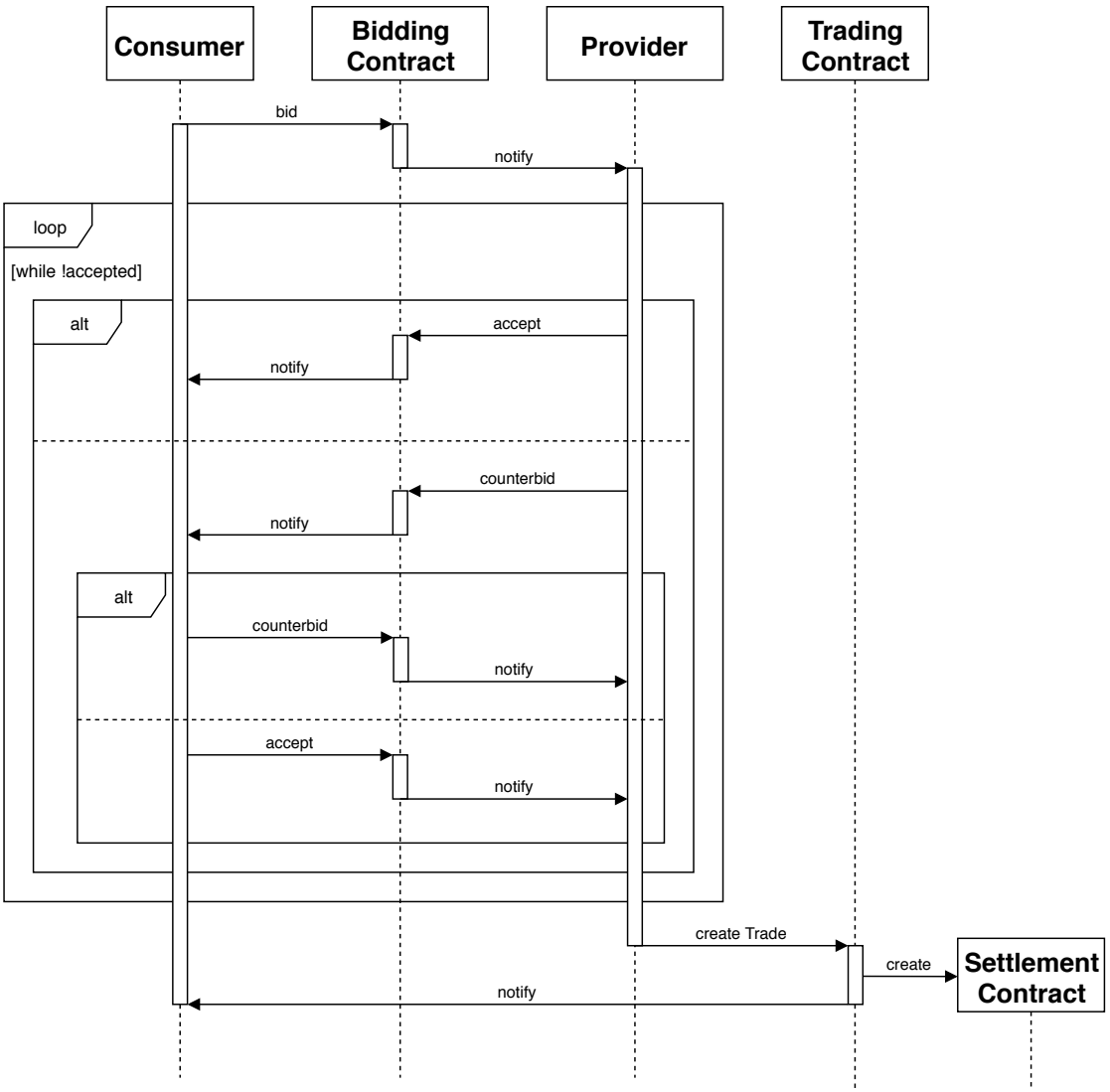

Figure 6.6: Negotiation between provider and consumer

After the creation of the bidding contract, both parties will be able to use this contract to negotiate about the terms of the trade (see Figure 6.6). The consumer uses the smart contract to submit the first bid. The provider is informed about the bid and has two ways to respond to it. Should the provider be satisfied with the bid, it can accept the offer directly. If there is any further need for negotiation, the provider can submit a counter-bid, via which the consumer will also be informed. The consumer now has the same options as the provider to respond to the counter-bid. Either the consumer accepts the counter-bid or creates a new counter-bid. This process can be repeated until the two parties have agreed on the terms of the data trade. As soon as an offer is accepted, the provider takes over the creation of the trade through the trading contract.

Here, all conditions of the trade are determined and who is involved in it. Creating the trade also generates a new settlement contract which is used for the settlement of the trade (see Section 6.2.7). After the creation of the trade, both provider and consumer are notified, so that they can continue with the data trading process.

If the consumer wants to buy the product directly, no negotiation is necessary. In this case, the trading contract offers the possibility to create a trading request for a specific product, which can be accepted or declined by the provider. If the provider accepts the request, a trade will be created with the price specified in the product contract. The advantage here is that no additional fees arise from a negotiation.

6.2.6 Routing and Transmission

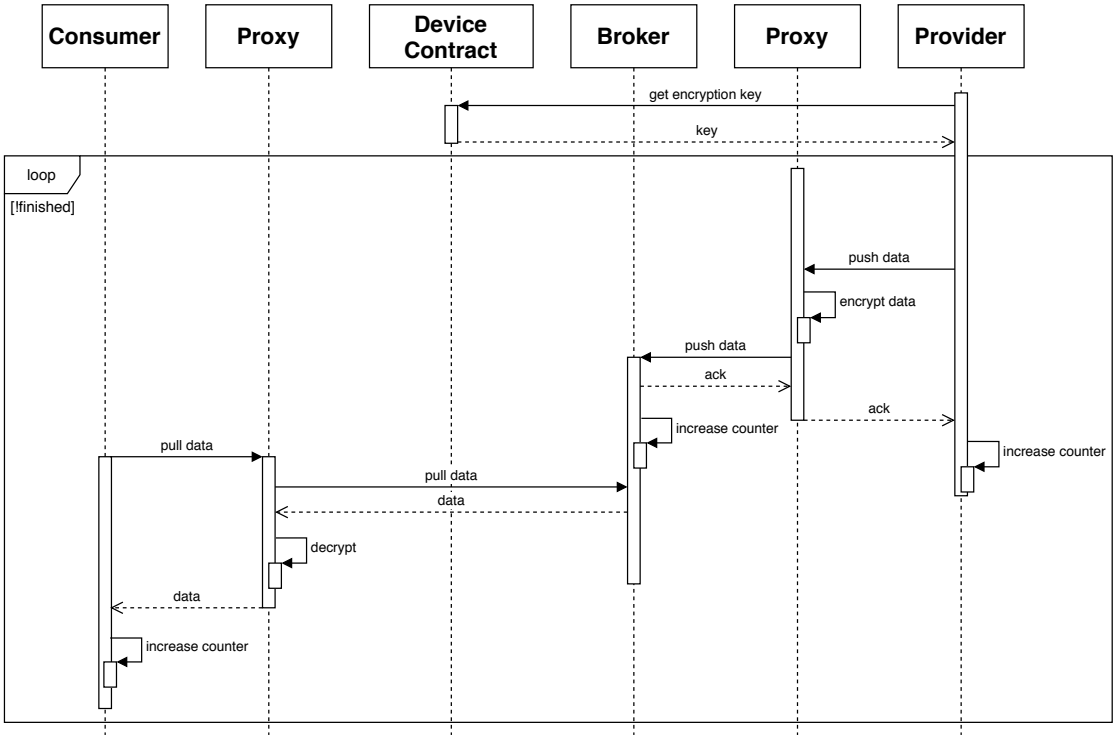

Figure 6.7: Routing and transmission sequence

As soon as the consumer has paid for the product, the provider receives a notification to start the data transfer (see Figure 6.7). In order to encrypt the data to preserve integrity, confidentiality and privacy, the provider needs to get the consumer's public key from the device contract. After that, the provider sends the data and the key to the proxy. The data is encrypted with the key and sent to the responsible broker. After the broker has confirmed that it has received the data, the provider increments its counter which records the number of transmitted data packets. The broker also manages a counter, which indicates how much data has been received. After paying for

the product (see Section 6.2.7), the consumer starts to pull the data from the broker. The data is also routed through the proxy, which decrypts the data and sends it to the consumer. After receiving the data, the consumer also increases its counter. This process is repeated until the trade expires, which is determined at the time of purchase. If the consumer desires an extension, it must renegotiate with the provider.

6.2.7 Settlement

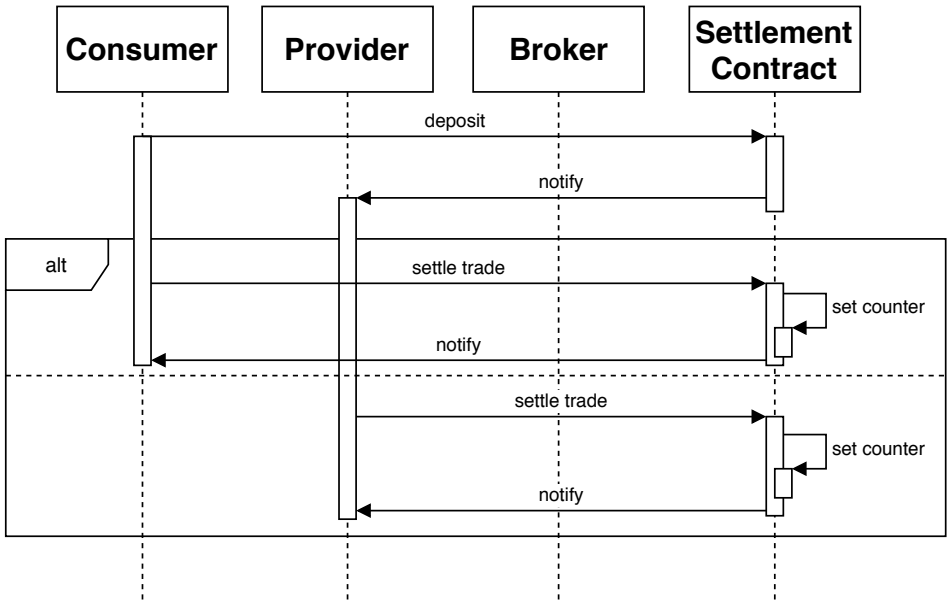

Figure 6.8: First stage of the settlement process

The settlement process (see Figure 6.8) is started before the data is actually exchanged. This starts with the payment of the product by the consumer. The reason for this is to protect the provider from sending the data to a consumer who is actually not able to pay for the product. On the other hand, the consumer is also protected against malicious providers, since the settlement contract acts as a conditional escrow, which ensures that the funds are only transferred if certain conditions are met.

Providers, consumers and brokers monitor how many data packets are transmitted, which becomes clear in the discussion of how the data is delivered (see Section 6.2.6) from the provider to the consumer. After the data transmission, the provider and consumer send a transaction to the settlement contract to settle the trade by disclosing their counters. These counters are used by the settlement contract as a basis for deciding if a dispute should be lodged after receiving the last counter (see Figure 6.9). If the counters match, the funds are transferred to the involved parties. Otherwise the broker will be informed of a dispute, which will then react to it by sending a transaction with its counter to the settlement contract, which will dissolve the dispute and then transfer the money.

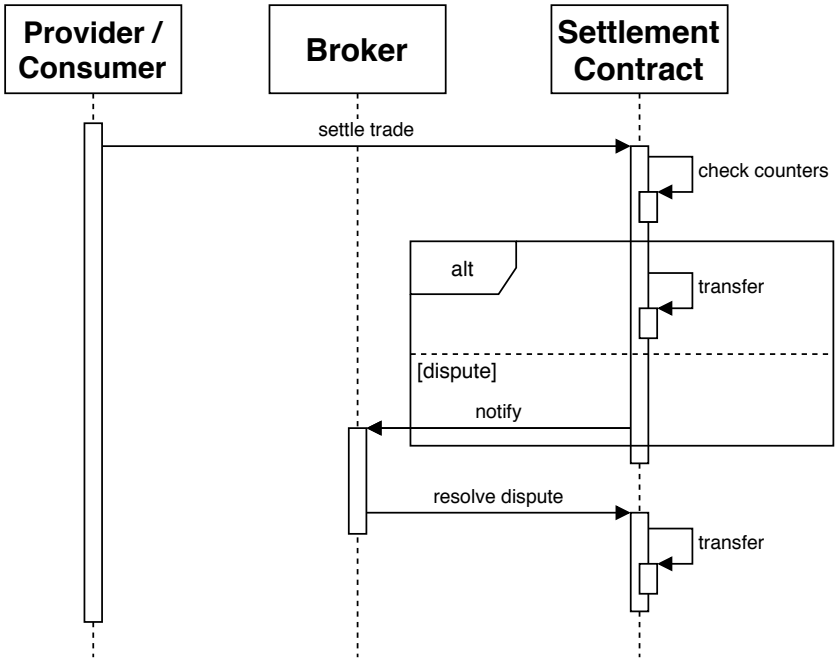

Figure 6.9: Second stage of the settlement process

6.2.8 Rating

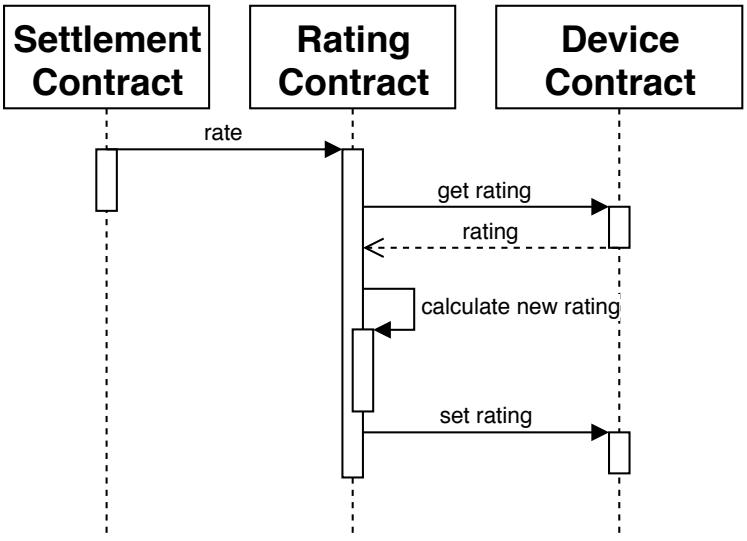

Figure 6.10: Rating sequence for a data trade

The data marketplace uses a rating system to reward honest and to penalize misbehaving participants. The rating process (see Figure 6.10) is initiated by a settlement contract which calls the rating contract. Hereby, the rating contract ensures that a settlement contract can only rate the participants which were involved in the corresponding trade. After the contract has ensured that the settlement contract is authorized to carry out a rating, it asks the device contract for the current rating of the provider or consumer. The rating is calculated and then set by using the device or broker contract.

CHAPTER 7

Implementation

In this chapter, we examine how exactly the framework for the decentralized data marketplace is implemented. For this, we discuss the technologies used as well as the implementation of the individual components. At the beginning (see Section 7.1) we talk about the implementation of the smart contracts, in particular Create, Read, Update, and Delete (CRUD) operations, authentication and authorization and the process of data trading. This is followed by an examination of the implementation of the individual services (proxy and broker) that are used to enable data trading for the IoT devices (see Section 7.2). Finally, in Section 7.3, we look at the implementation of the graphical user interfaces.

7.1 Smart Contracts

In Section 4.5, we have already taken a closer look at several smart contract and decentralized application platforms, which made decision-making easier when choosing the right technology for the implementation. Ethereum was able to differentiate itself from other platforms thanks to the existing tools and support of the community. Furthermore, Ethereum is a permissionless network, which is necessary for a decentralized data marketplace that should be accessible to everyone. Some of the tools provided for the development, testing and deployment of smart contracts on the Ethereum platform are part of the Truffle Suite[1] which was very heavily used for the implementation of the decentralized application. It allows management of smart contract artifacts, automatic testing of smart contracts, easier deployment through scripts, execution of external scripts and more useful features that made the whole implementation process easier. With Ganache, it also provides a local blockchain that can be used with little effort, which removes the effort of creating a testing network.

[1] https://www.trufflesuite.com/

7.1.1 CRUD Operations

The main task of the user, device, broker and product contract is to provide CRUD operations for the respective entities that are to be saved via the smart contracts. But also the negotiation and trading contract must at least provide create and read operations. Solidity provides two data structures for storing collections. The first option would be to save the data in an array of fixed or dynamic length. However, the data can only be accessed through the index, which is sometimes not sufficient. The other option would be to save the data in a mapping, which enables random access through a key.

Since a mapping also enables random access, it offers the opportunity to efficiently read specific entries from the smart contract, which is an essential functionality for the data marketplace. However, this advantage also comes with the disadvantages that Solidity does not offer the possibility to iterate over the keys and entries of mappings and there is no possibility to determine the number of entries in a mapping. To compensate for this, we store an index for the mapping, which is a dynamic array that stores all keys of the mapping separately. The number of entries in the array can be easily determined and thus also the number of entries in the mapping. It is also possible to iterate over arrays, which indirectly makes it possible to iterate over the mapping, since the keys of the mapping that are stored in the array can be used to directly access the entries of the mapping. The combination of a mapping and an array therefore offers the basic requirements for managing the data.

Creating a new user, device, broker or any other entity that needs to be persisted through a smart contract, requires the smart contract only to store the entity in the mapping with its ID as the key and adding the key to the index. Furthermore, the necessary references to other entities have to be set by calling the corresponding function of the other smart contract. For example, when creating a device, the ID of the device must be added to the user, which is saved via the user contract. The smart contracts provide the necessary functions to set these relationships between entities. When updating an entity, the smart contract first performs a look-up with the key and then updates the entity. With find by id, find by index and count functions a smart contract provides the necessary interface to retrieve specific entries or iterate over all entries stored in the mapping of the smart contract. To iterate over all entries, the caller gets the size of the index array and retrieves each entry one by one by providing the index. Deleting entries is only possible for users, brokers, devices and products whereby only a soft delete is performed by setting a deleted flag. Furthermore, delete operations are cascaded, for example deleting a user means that the relevant brokers, devices and products are also deleted.

7.1.2 Authentication and Authorization

Authentication and authorization is an important factor that was taken into account when implementing the smart contracts, as these are public and accessible to everyone. Therefore, for all functions of the smart contracts that change the state, the sender's address is first used to check whether the account is authorized to carry out this action or it is ensured directly via the code that an account can only manage its own data. It was also necessary to consider how the smart contracts are linked to each other dynamically and during deployment. The addresses of settlement contracts, for example, must also be saved in the rating contract so that it can ensure that only these contracts can change the rating. It is not possible to replace the implementations of the smart contracts after the deployment because the addresses can only be set once in the corresponding contracts. This has the advantage that no central authority can swap the implementation in its favor after the deployment, but it is also not possible to carry out an update.

7.1.3 Data Trading

In Section 6.2, we have already discussed smart contract based data trading. Now it is time to take a closer look at the implementation of the individual components which are necessary for this.

The negotiation contract stores all negotiation requests and negotiations. To request a negotiation, the consumer only needs to call the smart contract's request negotiation function, specifying the ID of the product. In contrast, the provider can use the accept negotiation function of the smart contract to start the negotiation process. When the function is called, a negotiation is saved on the blockchain and a new bidding contract is deployed (see Figure 6.5). The negotiation object specifies the consumer, the bidding contract and the subject of the negotiation, i.e., the product.

After that, the consumer can then proceed with creating a bid through the previously deployed bidding contract and call the make bid function stating the bid which includes the price, as well as the start and end time of the data stream. The same is also possible for the provider, so that offers can be exchanged (see Figure 6.6). It further provides functions to accept and cancel the bidding process, which ensures that no new actions are possible apart from retrieving data from the smart contract. The smart contract regulates that both parties can only alternately call functions of the smart contract. The last saved bid of a bidding contract in which the accept function was called thus represents the offer that has been agreed.

Purchase without negotiation takes place directly via the trading contract by calling the request trading function, whereby the consumer specifies which product he wants to buy, the broker which facilitates the trade and the start and end time of the data stream. The request is then stored on the blockchain whereby the consumer automatically accepts the specified price by the provider.

Table 7.1: Stored trades

Provider	Consumer	Broker	Product	Start	End	Cost	Freq	Settlement
0x9395...	0x1b72...	0xb3d2...	4	1587981426	1588010400	5	10	0x1b72...
0xb3d7...	0x3195...	0xC605...	4	1588032000	1588096800	8	5	0x49Bf...

Subsequently, the provider has two options to create a trade. For this, it can either call the accept trading request function or create function of the trading contract. In the first variant, the provider must specify the trading request which should be accepted, whereby the trading request stores the conditions of the trade. In the second variant, the provider specifies the negotiation and the broker, whereby the conditions agreed on in the bidding contract are used for the trade. In both cases, the smart contract enforces that a trade can only be created with the conditions that both partners have agreed on. When a trade is created, it is saved in the trading contract and stores the conditions of the trade and the address of the settlement contract that is created in the same move (see Table 7.1).

The settlement contract provides the consumer with a deposit function to send Ether to the contract and ensures that a correspondingly high amount is paid in order to cover the costs of the trade. Then, both provider and consumer can start exchanging data. After the transfer has been completed, both provider and consumer call the settle trade function of the contract and provide their counters. When the function is first called, only the caller's counter is saved (see Figure 6.8) and the second call compares the two counters and continues with the settlement (see Figure 6.9). If both counters match, the counter is used to calculate how much money is paid out to the provider, broker and consumer. Then the funds are transferred to the corresponding parties and the trade is considered settled.

However, since certain problems can arise, such as that the provider and consumer disagree or do not continue the process, certain exceptions must be considered. If the counters do not match, a dispute is determined which the broker must resolve. For this, the smart contract offers the settle dispute function which can only be used by the broker after a dispute has occurred, whereby the broker determines the value of the counter. Since it cannot be ruled out that one of the two parties does not call the settle trade function, a resolve timeout function has been added to the settlement contract. If one of the parties does not call the settle trade function up to a certain point in time (24 hours after the actual end of the trade), the party that behaved as expected can withdraw the funds from the contract. If the settle trade function is not called at all, this will send everything back to the consumer.

At the end of the settlement process, the settlement contract calls the rating contract to evaluate the involved parties (see Figure 6.10). The rating process is kept very simple since the focus is more on ensuring that only the settlement contract is authorized to rate a user as has already been discussed in Section 7.1.2. Nevertheless, the contract provides two functions for the settlement contract to rate the involved parties of a trade. One calculates and sets the rating in the case of a successful settlement and the other

one if a dispute occurred. In the first case, the user's rating increases by one, whereby in the second case it decreases by five. A user's rating is represented by a numerical value between zero and 100, with 100 being the best and zero the worst. In future work, more complex calculations of the rating could be considered, but currently Solidity does not support fixed point numbers, which makes complex calculations difficult.

7.2 Services

Both the proxy and the broker were implemented in the Go[2] programming language. During implementation, particular attention was paid to maintainability and interchangeability. This makes it possible to react to changing requirements and enable further development of the system. This is reflected in the selection of the technologies used and the architecture of the applications.

The applications enable Inter Process Communication (IPC) through the usage of gRPC Remote Procedure Calls (gRPC). Proxy and broker provide multiple gRPC services, which are described through service definition files written in the protocol buffer language. The protocol compiler generates the source code for client and service stubs, whereby several programming languages are supported. Providers and consumers can thus use a programming language of their choice, provided that it is supported by Protocol Buffers.

Both services must be able to communicate with the smart contracts deployed on the Ethereum platform. Ethereum provides an API that can be addressed via JSON-Remote Procedure Calls (RPC) as well as the matching client written in Go. However, a lot of boilerplate code would have had to be written to be able to use all smart contracts. For this reason, a Go binding generator was used that generates the corresponding counterparts in Go given the source code files of the smart contracts. Ethereum smart contracts can only return multiple values and no structs, which means that they are only returned in an anonymous struct by the contract bindings. However, in order to be able to work with the structs as they are also used in the smart contracts, an adapter was created for each contract binding, which wraps the individual variables in the corresponding struct.

All parts of the two applications are decoupled from each other by interfaces. This offers the advantage that individual parts can be easily replaced or extended. This can be very useful, especially with the API, if at some point it turns out that gRPC is not particularly suitable and another technology has to be used.

[2] https://golang.org/

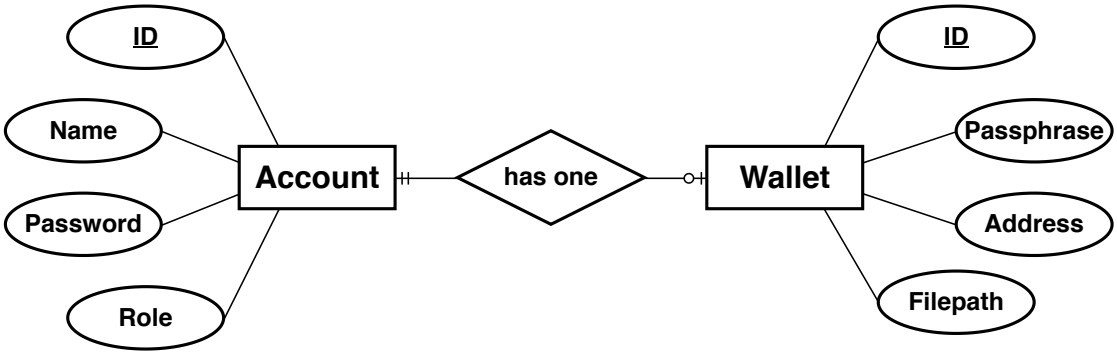

Figure 7.1: Entity–relationship model for the proxy

7.2.1 Proxy

The proxy implements 13 gRPC services. These services include one service for each smart contract, which enable the communication with the smart contracts deployed on the blockchain using a proxy account with its corresponding wallet. The accounts with their wallets can be managed by an administrator through the account service and the wallet service. Furthermore, the proxy provides a discovery service through which brokers and products can be searched. An authentication service enables the users of the proxy to authenticate themselves and a message service is responsible for the encrypted transmission of the traded data.

The data of the accounts and wallets are stored in an SQLite database (see Figure 7.1). An account consists of a username, password and role (admin or user), whereby bcrypt is used to only store the hash value of the password in the database. A wallet can be added to each account, with one wallet consisting of the Ethereum address, public key, passphrase and the path to the key file. The key files are stored in a separate folder on disk, whereby every file is encrypted with the passphrase specified for the wallet.

After accounts have been created for the users, they can authenticate themselves using their username and password and receive a JSON Web Token (JWT) in exchange. In addition to the standard claims, the user ID and the role of the user are also encoded in the token. All services except the authentication service require that a valid JWT is present in the authorization header of every request.

Calling up smart contracts with the help of contract services works for every smart contract according to the same principle (see Figure 7.2). The provider or consumer issues a unary call to the corresponding contract service with the JWT included in the header. The proxy selects the appropriate wallet based on the provided token and unlocks the key file. After that, a transaction signer is created which is passed to the contract bindings that use it to create the transaction. After sending the transaction to the smart contract, the account gets locked again. This enables the unencrypted key to be kept in memory only as long as necessary to create the transaction. Events can also be listened to via the various contract services that are streamed to the caller.

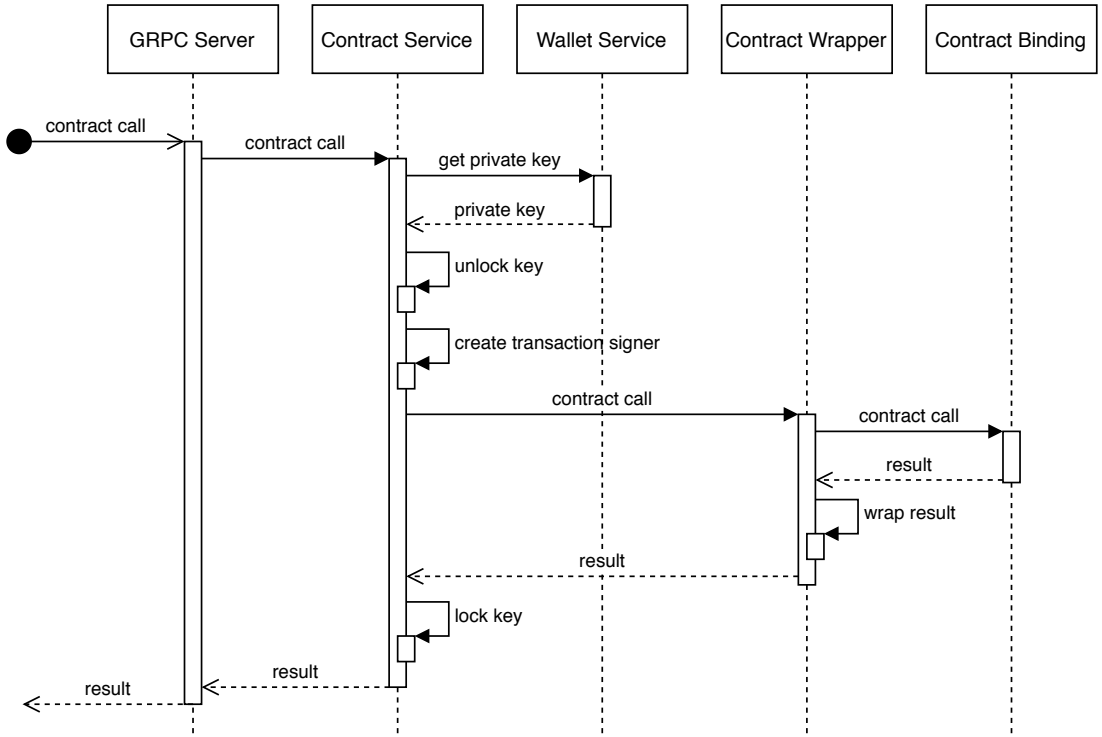

Figure 7.2: Sequence of interactions during a contract call through the proxy

The discovery service provides the user with the possibility to search for brokers and products by providing a search query (see Figure 6.4). Brokers can be filtered by name and locations and products by data type, frequency and cost. The blockchain is scanned for brokers directly by the proxy, whereas a product query is forwarded to a broker that was specified by the user in the search query. In both cases, the filtered results are streamed to the caller.

Messages that are to be transferred from providers to consumers are encrypted using the crypto message service and forwarded to the specified broker. The public or private key of the respective Ethereum account is used to encrypt and decrypt the data using the Elliptic Curve Integrated Encryption Scheme (ECIES). This means that no additional key pair has to be created and no key exchange has to take place, since the public key of the consumer can be read directly from the device contract. The messages are pushed unencrypted to the proxy, which uses the public key of the consumer to encrypt the payload. The consumer pulls the message through a proxy, which again uses the token to determine the appropriate key file and uses the account's private key to decrypt the payload (see Figure 6.7).

7.2.2 Broker

The broker implements three services and makes two of them available via gRPC. These three services include the message service, the dispute service and the discovery service. The broker also has its own Ethereum account so that smart contracts can be addressed, which is necessary for dispute resolution.

The main task of the broker is fulfilled by the message service, which accepts and counts messages and puts them in a queue, where they can then be pulled by the consumer. This enables the consumer to be offline while the provider sends the messages, but only a certain number of messages are stored in the queue to limit memory usage and the broker also does not provide a persistent storage. Furthermore, the prototype does not provide any means of authentication, which enables anyone to consume the messages from the broker. In future work, the broker would have to be expanded to include an authentication mechanism, which ensures that only the consumer who bought the data can pull the messages from the server. For example, token-based authentication could also be implemented here, whereby the user does not provide username and password to receive a token, but has to prove that it has the private key of a certain Ethereum account.

Like the proxy, the broker takes over the task of providing different products by entering a search query. As has already been mentioned in Section 7.2.1 the search query enables filtering by data type, frequency and cost. The broker scans all products saved in the product contract and streams the filtered results to the caller, which in most cases is a proxy that forwards the results to the consumer.

The last service is responsible for dispute resolution. In the event that providers and consumers cannot agree during the settlement process, a dispute event is triggered on the blockchain. The broker listens for these events and sends a transaction to the corresponding settlement contract (see Figure 6.9) to resolve the dispute by providing the message count recorded by the message service.

7.3 Graphical User Interfaces

Using Angular[3], a TypeScript-based web application framework, two independent graphical user interfaces were implemented. Furthermore, both user interfaces were created using material design[4]. Angular applications can be composed of multiple modules, whereby each module serves its own purpose which contributes to interchangeability and maintainability. For this reason, care was taken during the implementation that the applications are broken down into several modules. One of the two user interfaces is responsible for the administration of the proxy, whereas the second user interface gives the user the opportunity to communicate with the smart contracts. In combination, both interfaces allow the user to perform all of the tasks necessary to use the data marketplace.

[3] https://angular.io/
[4] https://material.io/design/

7.3.1 Proxy

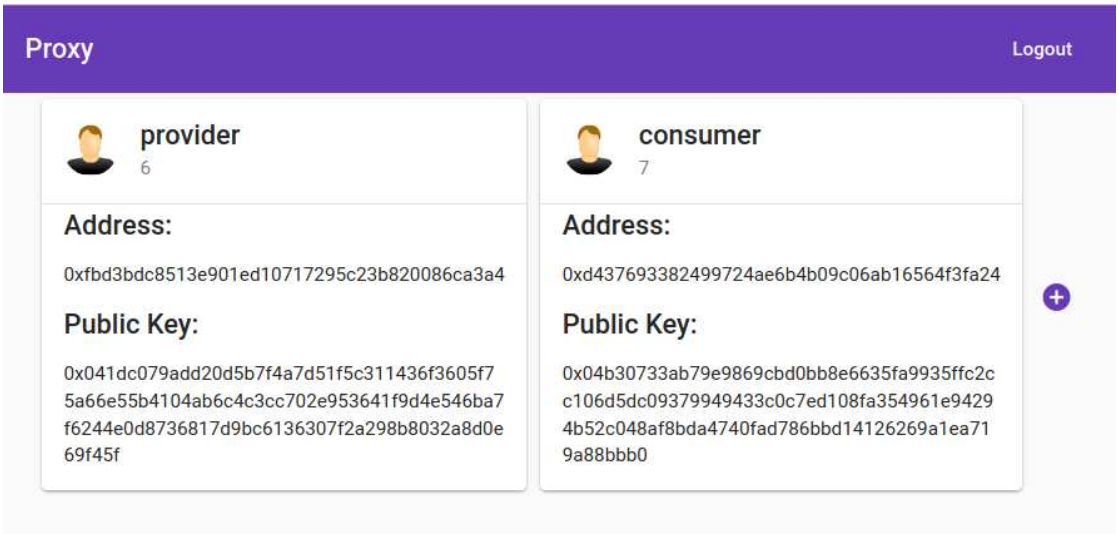

Figure 7.3: Graphical user interface of the proxy

The user interface of the proxy was kept very simple and only contains the necessary functions to enable the administrator to create new accounts with wallets for devices that should communicate with the proxy (see Figure 7.3). Two modules provide the necessary functionalities for this.

As mentioned in Section 7.2.1, the proxy provides multiple services that can be addressed via gRPC. To enable the communication between the GUI and the proxy, gRPC-Web is used to generate the client service stubs and create the client code. At the moment, it is unfortunately not possible to use gRPC-Web in the browser without further ado, which requires the use of a proxy that accepts Hypertext Transfer Protocol (HTTP)/1.1 requests and translates them into HTTP/2 requests (see Figure 7.4). For this, we configured an Envoy proxy, which has good support for gRPC as well as HTTP/2 and is mentioned in the documentation of gRPC.

In order to use the proxy, the administrator has to authenticate itself. For this purpose, the authentication module provides the user with a login component, which is used to send the credentials to the authentication service of the proxy. If authentication is successful, an access token will be made available which will be sent to the proxy with every future request.

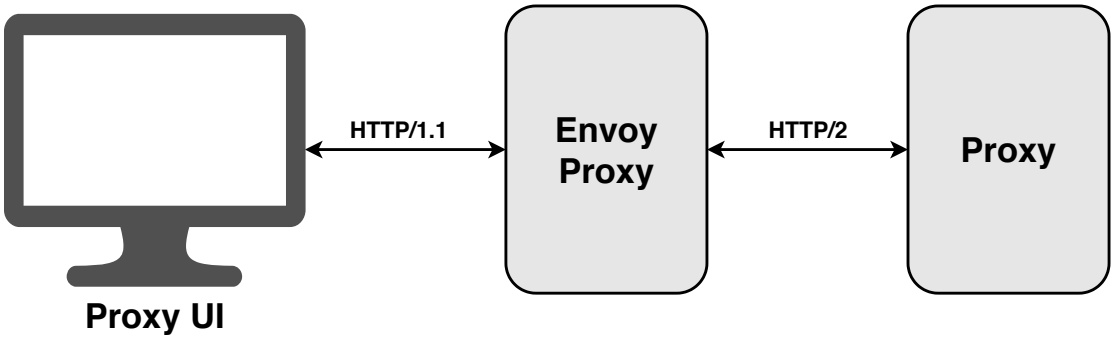

Figure 7.4: Communication from GUI and proxy through envoy proxy

The account module provides different components in order to view the accounts managed by the proxy and to create new accounts. When creating an account, the administrator specifies the user name, password of the account and the passphrase of the wallet with which it is to be encrypted. The GUI then sends the requests to the account and wallet service of the proxy. For the time being, more functionalities such as updating and deleting accounts have not been implemented. Furthermore, the administrator lacks the options to access the wallets of the different accounts, which should be considered in future work.

7.3.2 Smart Contracts

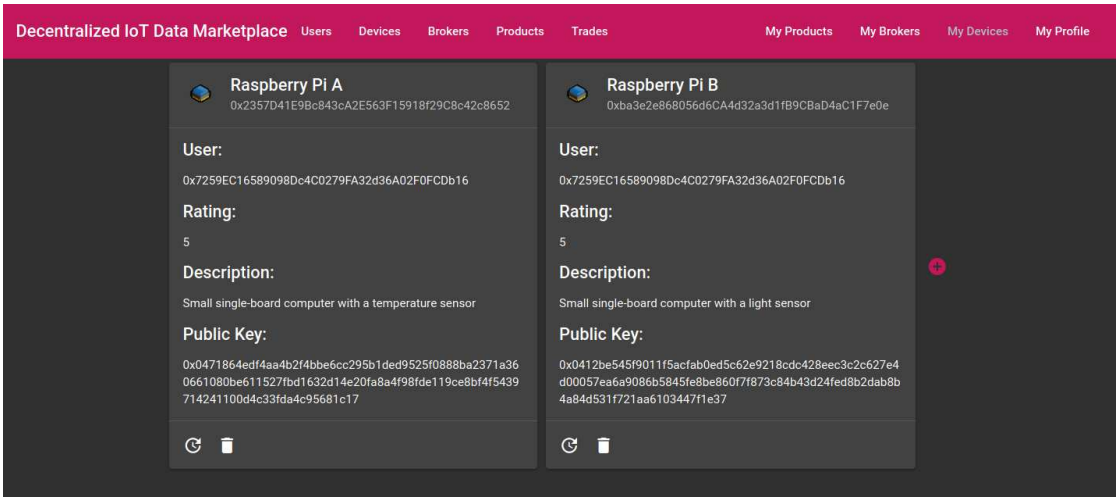

Figure 7.5: Graphical user interface for the smart contracts of the data marketplace

The graphical user interface (see Figure 7.5) for the decentralized application covers two major tasks. It enables the user to view the data stored on the blockchain and to manage their own data. The user can view all user profiles, devices, brokers, products and trades. All data marked as deleted is filtered out by the application. If a user has saved their

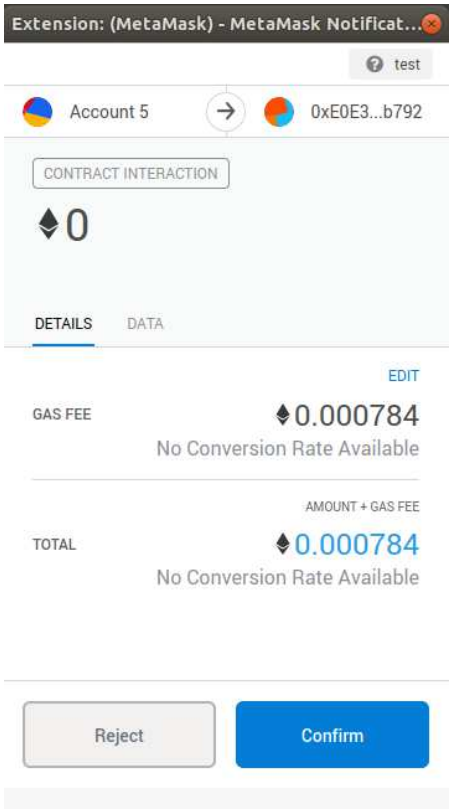

Figure 7.6: MetaMask pop-up to review the transaction

own user profile, devices, brokers or products on the blockchain, they can view, edit or delete them using the respective management view.

In order to be able to interact with the smart contracts, the GUI must be able to send transactions to the Ethereum network. For this purpose, we use MetaMask[5], which is a browser extension that provides wallet functionalities and connects the GUI with the Ethereum network. It further gives the user the possibility to review the transaction (see Figure 7.6) before it is send to the Ethereum network, by showing transaction details like the amount, gas price and gas limit.

As with the user interface for the proxy, this application is also divided into several modules: a user module, device module, broker module, product module and a trade module. Each module provides different components to view and manage the respective entity. Truffle contracts provide an abstraction layer which facilitates the interaction with the smart contracts by shadowing the complexities of the JavaScript Ethereum API.

[5]https://metamask.io/

The user interface still leaves some points open, which can be taken into account in future work. Negotiations were completely omitted in the prototype of the application, because we wanted to focus on more essential things in development. Furthermore, searching and filtering are not part of the prototype, however, they represent valuable functionalities that can still be added. Nevertheless, the user interface offers all the basic functionalities to get a deeper insight into the system and view the state of the blockchain.

CHAPTER 8

Evaluation

After the implementation of the decentralized IoT data marketplace has been discussed, it is particularly interesting to examine what costs arise from the use of the implemented smart contracts. For this, we first look at the costs that arise from the deployment of the smart contracts (see Section 8.1.1), followed by the costs for the administration of users, brokers, devices and products (see Section 8.1.2). Finally, we consider the costs incurred by the individual participants during the data trading process (see Section 8.1.3). In addition to the costs, we also discuss how smart contracts are suitable for implementing a decentralized data marketplace (see Section 8.2). To do this, we consider the limitations and possible weaknesses of smart contracts, which were observed during implementation.

8.1 Costs

In Section 4.2, we discussed the concept of gas and payments, which Ethereum uses to protect the platform from misuse. This results in some costs for the use of smart contracts, which are examined in more detail in order to be able to estimate when such use is worthwhile. For this, we consider the costs that initially arise from the deployment of the smart contracts, followed by the costs from the administration of the user profile, the brokers, devices and products and finally the costs that arise during the data trading process.

In order to not only describe the costs in the form of gas, we can also specify the value in a specific fiat currency. All that is required is the used gas, the gas price used for the transactions and the current exchange rate from ETH to the desired currency. It should be noted that the gas price is determined by the sender of the transaction. However, if the gas price is too low, the transaction may never be included in a block or much later, since miners tend to include transactions with higher transaction fees. As of May 17, 2020, the price for one Ether is $207.08 and we are assuming a gas price of 20 gWei.

8.1.1 Deployment Costs

Table 8.1: Smart contract deployment costs

Smart Contract	Gas	Ether	USD
Migrations	190982	0.00381964	0.79
UserContract	1468608	0.02937216	6.08
DeviceContract	3938622	0.07877244	16.31
ProductContract	3054637	0.06109274	12.65
BrokerContract	1939980	0.0387996	8.03
NegotiationContract	1793263	0.03586526	7.42
TradingContract	3264283	0.06528566	13.51
RatingContract	437819	0.00875638	1.81

The first costs arise when deploying the various smart contracts for the decentralized IoT data marketplace. However, it should be noted here that these costs, with the exception of the settlement and the bidding contract, only have to be paid once by the operator so that the data marketplace can be put into operation. By adding up the costs from all contract deployments (see Table 8.1) we get a sum of 16088194 gas or $66.63 for the deployment of all smart contracts. It may appear that there are a lot of costs associated with deployment, but it must be considered that, contrary to a centralized approach, the operator does not have to pay running costs for infrastructure and operations.

8.1.2 Costs of Administrative Functions

After the costs for the deployment of the smart contracts have been clarified, it is necessary to examine the recurring costs that arise from calling up the various functions of the smart contracts. However, functions that do not change the state of the blockchain can be excluded from these considerations, since these are carried out by a local node and therefore there are no costs for the caller. This means that all participants in the network can read out all the data that is stored by the smart contracts of the data marketplace without having to pay transaction fees.

In contrast, all operations that change the state of the blockchain, consume gas. This includes, among other things, all administrative functions, such as creating, updating and deleting users, devices, brokers and products. In Section 7.1, we have already discussed how the CRUD functionalities of the various smart contracts are implemented. By making sure that no loops are used during the implementation, the costs for these operations (see Table 8.2) remain constant even if the number of saved entries increases as the examples from Figure 8.1 show. However, it must be noted that this only applies to delete operations if they are not cascaded.

Table 8.2: Costs of Create, Update and Delete Operations

Smart Contract	Create			Update			Delete		
	Gas	Ether	USD	Gas	Ether	USD	Gas	Ether	USD
UserContract	178103	0.00356206	0.73	72994	0.00145988	0.30	45171	0.00090342	0.18
DeviceContract	252964	0.00505928	1.04	70690	0.0014138	0.29	46697	0.00093394	0.19
ProductContract	300232	0.00600464	1.24	83446	0.00166892	0.34	49075	0.0009815	0.20
BrokerContract	207946	0.00415892	0.86	55757	0.00111514	0.23	46377	0.00092754	0.19

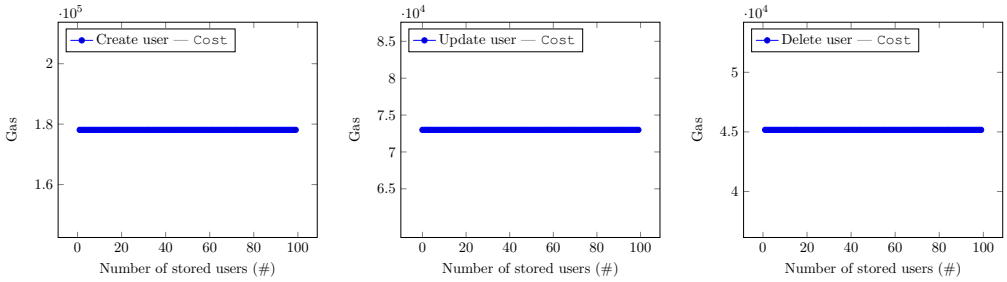

Figure 8.1: Costs of user create, update and delete operations with increasing number of stored entries

Although these costs remain constant, they are not the same for all entities. One reason for this is that not every entry requires the same storage space. For the illustration in Figure 8.1, we used test data, which requires the same storage space for every entry. In reality, however, the memory consumption depends on the data that the user of the smart contract wants to save. A good example for this is the description of a device which can vary greatly in length. With some devices, only a very short description may be saved, whereas users can also save very long descriptions, which impacts the used gas.

In order to be able to estimate how strong the influence of the storage space is on the costs, several devices were created and updated, whereby the length of the description was continuously increased. As can be seen in Figure 8.2, the costs rise in very small steps. This is due to the fact that the size of the array, which stores the description of a device, is always dynamically increased by a certain value, whereby the costs thereafter remain constant for the period in which the allocated memory of the array does not have to be changed. It can thus be stated that all attributes of variable length significantly influence the costs for saving and updating an entry.

Since we have so far limited ourselves to the costs of create and update operations, we will now look at the costs of delete operations. The costs of delete operations have already been stated in Table 8.2. As mentioned before, however, this only applies if there is no cascading deletion process. This is only the case if the entity does not reference any other entity. In most cases, however, this is not the case. For example, a device may have multiple products that it references. If this device were now deleted, the products to which it refers would also be deleted.

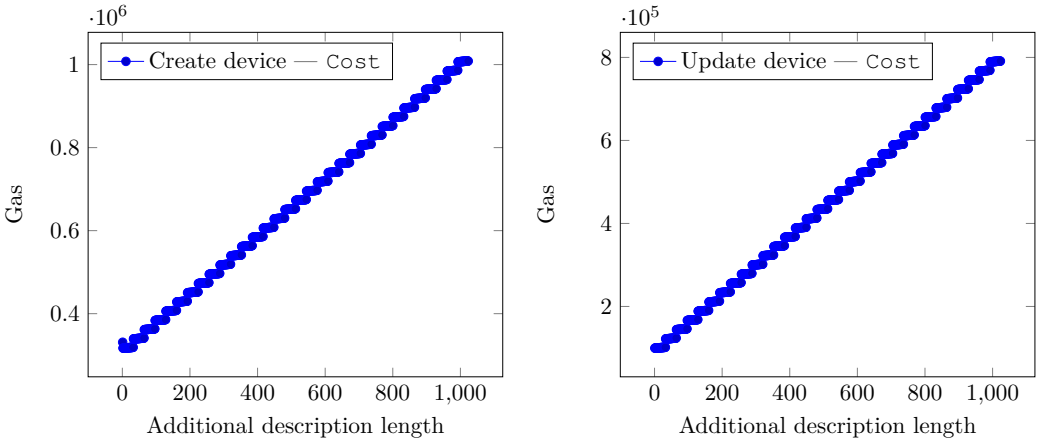

Figure 8.2: Costs of device create and update operations with increasing memory consumption

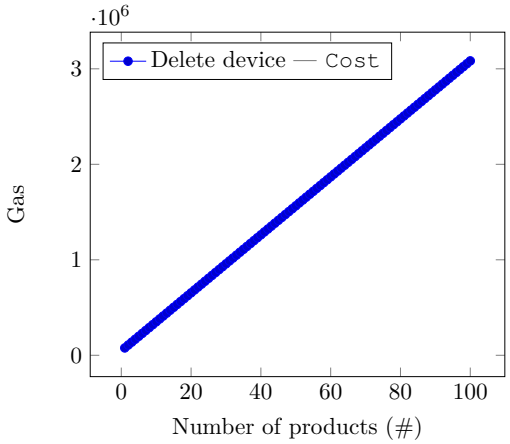

Figure 8.3: Cost of cascading device deletion

With the deletion of an entity that triggers the deletion of another entity, the costs increase with the number of entities to which it references. To test how this affects the costs (see Figure 8.3), we created devices with a different number of products and deleted them. It can be seen that the costs increase linearly with the number of products of the device. Another possibility how the deletion of a device could have been implemented would be that all products have to be deleted individually before the device can be deleted. However, this incurs much higher costs because the transaction fees must be paid for each deletion. For each additional product, the used gas increases by 30365, whereas the deletion of a single product would consume 49075 gas. The problem with cascading deletion, however is, that entities may refer to too many other entities and the used gas exceeds the gas limit at some point, causing the transaction to fail.

Table 8.3: Data trading costs

Smart Contract	Operation	Gas	Ether	USD
NegotiationContract	requestNegotiation	213587	0.00427174	0.88
NegotiationContract	acceptNegotiationRequest	589352	0.01178704	2.44
BiddingContract	makeBid	89915	0.0017983	0.37
BiddingContract	accept	28944	0.00057888	0.11
TradingContract	requestTrading	326777	0.00653554	1.35
TradingContract	acceptTradingRequest	1051960	0.0210392	4.35
TradingContract	create	1198142	0.02396284	4.96
SettlementContract	deposit	23303	0.00046606	0.09
SettlementContract	settleTrade	65742 180951	0.00131484 0.00361902	0.27 0.74

8.1.3 Data Trading Costs

In addition to the costs incurred for the administration of the user profile, brokers, devices and products, the costs that occur in data trading are particularly interesting. For this we look at the costs of the individual functions (see Table 8.3) that are carried out in the course of the data trading process. The costs of these operations remain constant except for the acceptance of negotiation and trading requests. These depend on the number of requests that the provider has not yet processed, because if such a request is accepted, all pending requests must be iterated over to remove it. We differentiate which costs are incurred by the provider and which are incurred by the consumer. Furthermore, the costs are also heavily dependent on whether the terms of the trade are negotiated first or whether a direct purchase request is made. With the latter, some costs can be removed because all interactions with the negotiation contract, as well as the deployment of a bidding contract and its use are completely eliminated.

First, we look at the costs involved in trading without negotiation. The consumer uses the following functions: requestTrading, deposit and settleTrade. The provider uses acceptTradingRequest and settleTrade. By calling acceptTradingRequest, the provider directly deploys a settlement contract, which results in significantly higher costs for the provider. The settleTrade function is used by both parties and depending on who calls the function first or second, different costs arise because other operations are carried out (see Section 7.1.3). In our example, we assume that the provider calls the settleTrade function first. If we sum up the costs of carrying out the individual operations, we get a value of 1117702 gas or $4.63 for the provider and 531031 gas or $2.2 for the consumer.

If a negotiation takes place before the actual trade, the costs for this must also be considered. In this case, the provider uses additional functions such as acceptNegotiationRequest and optionally the functions makeBid and accept. It also applies to the consumer which carries out the functions requestNegotiation, makeBid and optionally accept. The sequence of the negotiation process has been discussed in Section 6.2.5 and

provides more information about which operations are performed by whom and how often. In our investigation, we restrict ourselves to an exchange of bids, whereby only the consumer makes a bid that is accepted directly by the provider, which is the shortest possible negotiation process. Furthermore, no trading requests are required and the trade is created by the consumer directly using the create function, specifying the negotiation.

Summing up the costs for the provider and consumer we get 507756 gas or $2.1 for the consumer and 1882180 gas or $7.79 for the provider. When comparing these costs to the costs that arise without negotiation, it can be seen that these are 4.38 % lower for the consumer and 68.39 % higher for the provider. The reason for this is that a trading request requires more storage space than a negotiation request and therefore causes higher costs for the consumer. However, this does not apply to the provider who bears a considerably larger part of the costs than the consumer. This is due to the execution of the acceptNegotiationRequest function, since a new bidding contract is created when it gets executed.

8.2 Critical Discussion

After we have discussed the costs of using smart contracts, we want to take a closer look at how smart contracts are suitable for implementing a decentralized data marketplace for the IoT. For this we consider not only the advantages of smart contracts and blockchain technology, but also the disadvantages that were mainly perceived during the implementation.

Among other things, we use smart contracts to store various entities on the blockchain, whereby the blockchain acts as a distributed database. In Chapter 2 we have already discussed the advantages of this approach, whereby the blockchain ensures transparency, integrity and verifiability of the data. In particular, the fact that transactions are either carried out completely or not at all, the integrity and consistency of the data which are stored by the smart contracts can be well guaranteed. However, there are disadvantages compared to traditional databases, which can be noticed when implementing and using smart contracts.

A disadvantage of smart contracts compared to traditional databases is that it is much more difficult to query the data. In contrast to smart contracts, traditional databases provide efficient query mechanisms for data retrieval. To query the data stored through the various smart contracts, we have to iterate over the entire storage of a smart contract and filter out the data that is of interest. It is of course possible to implement certain filter methods in the smart contract, but this causes additional costs.

Another problem that occurs is that Ethereum does not support structs as return values. This means that all attributes of an entity must be passed individually as return parameters. The stack also plays an important role here, which limits the number of local variables and thus the parameters that can be returned. For example, entities had to be split, or attributes had to be omitted to enable them to be returned. As a result, several function calls have to be made in order to be able to read out an entity completely.

During the implementation, we also had to cut back on the complexity of certain functionalities. A limitation of smart contracts is, for example, that they can only have a certain size. This can be a problem, especially with smart contracts that have to provide a variety of functions. While it makes sense to keep the smart contracts as small as possible, it is sometimes desirable to have certain functionalities encapsulated in one smart contract. In our case, the device contract provides a variety of functions related to devices that cannot be outsourced to another smart contract. We had to be particularly careful here, as the maximum size allowed was almost exceeded.

Furthermore, as has already been discussed in Chapter 7, smart contracts programmed in Solidity do not offer support for floating point numbers. One reason for this is the non-determinism of floating point operations. This is absolutely necessary when using smart contracts because each node must be able to achieve the same result, otherwise there is no consensus. However, floating point numbers are essential for more complex calculations and this hinders certain calculations that should be carried out with the help of smart contracts. In our case, we did not use a complex calculation method for the rating system and replaced it with a much simpler one, which is easy to implement with integers. Support for fixed point numbers is planned for the future, whereby, the presentation of the number is precisely defined.

In addition, the costs of using smart contracts are also a weakness. The users of the data marketplace have to decide for themselves which ratio between purchase price and data trading costs is suitable for them and when it is no longer worthwhile. However, other approaches could be followed which cause less costs. The negotiation process could take place completely off-chain, but then the advantages of blockchains would be lost. Furthermore, the administrative costs could be reduced by reducing the memory consumption. For example, the authors of [RRK18] use a DFS to store product information in JSON format.

The previous examples are not intended to present smart contracts as bad, but only to show a few negative features in addition to the desired positive ones, which we noticed during the implementation.

CHAPTER 9

Conclusion

The last chapter provides a summary of this work. For this summary (see Section 9.1) we take a recap of the previous chapters and their results to review the most important aspects of the work. Furthermore, we consider possible future work (see Section 9.2), which could be of particular interest for this field.

9.1 Summary

Blockchains and smart contracts are promising technologies for the IoT. Many use cases have already made this clear, including data trading. Data is a very valuable asset which can be traded over data marketplaces. They provide a platform for providers and consumers to enable the exchange of data. These data marketplaces enable organizations and individuals to offer their data free of charge or for a certain fee. Especially due to the constantly growing IoT, with more and more connected devices and the increasing amount of data being sent over the network, such platforms are becoming more and more interesting. Traditional data marketplaces come with different challenges concerning scalability issues, expensive infrastructure or single points of failure. This is where blockchain technology and smart contracts emerge, which offer the opportunity to create a decentralized IoT data marketplace.

In order to create such a decentralized IoT data marketplace, we started with literature work to look at related work and to gather the necessary background knowledge about the IoT, blockchains, smart contracts and data marketplaces. We were able to determine that some work already exists in relation to data trading and data marketplaces, both traditional approaches which are based on a centralized architecture and approaches that make use of blockchain technology. We identified several disadvantages of centralized data marketplaces, including scalability issues, expensive infrastructure, single point of failures, trust problems and privacy issues. In the case of blockchain-based data marketplaces,

however, we found out that they do not deal with all the important key elements of data marketplaces and rather focus on pure data trading.

Furthermore, in the course of this work, we also looked at different smart contract platforms and compared them with each other. For this, we have selected Bitcoin, Ethereum, Neo and Hyperledger Fabric. We compared the platforms with regard to smart contracts, programming languages, structure, openness, consensus mechanism, membership selection, energy consumption, support and tools. This gave us a deeper understanding of the different platforms and Ethereum turned out to be the best choice for the implementation of the prototype, because it enjoys the best support from the community and thus also provides many tools for developing, testing and deploying smart contracts.

We designed a framework for a decentralized IoT data marketplace, which is based on a three-tier architecture. Smart contracts provide various functionalities and enforce the rules of the data marketplace. A proxy gives providers and consumers the possibility to integrate IoT devices with little resources. A broker helps to facilitate the data trading process, takes over resource-intensive tasks and is responsible for dispute resolution in case a provider and consumer cannot agree during the settlement process.

Subsequently, we implemented the prototype. The smart contracts are developed for the Ethereum Platform in the programming language Solidity. We used the Truffle suite for the development, testing and deployment of the smart contracts. The implemented smart contracts use mappings to allow random access and we added an index array to enable the iteration over the stored entries. To secure the smart contracts, we check the sender's address of each transaction in order to determine if the account is authorized to carry out the operation. We implemented a negotiation and bidding contract to enable the provider and consumer to negotiate the terms of a trade. The negotiation contract stores all negotiations and the involved parties use the bidding contract to exchange offers. A trading contract enables direct purchase inquiries or trades to be created from a negotiation. Each trade is assigned a settlement contract, whereby the involved parties use the settlement contract to publish their counters to settle the trade. If they do not agree, the broker can use the settlement contract to resolve the dispute by publishing its counter.

After creating the prototype, we examined the costs of using the various smart contracts. We looked at the cost of deployment, administrative functions and the cost of data trading. We have determined that the costs remain constant for the most part and got a value of $4.63 for the provider and $2.2 for the consumer per data trade without negotiation. Last but not least, we discussed the problems encountered during the implementation when using smart contracts. These include inefficient query options, size limits of smart contracts, no structs as return values and missing support for floating point numbers. We found out that smart contracts are very well suited for the implementation of a decentralized IoT data marketplace, but it is also necessary to think about their limitations and disadvantages.

9.2 Future Work

In the course of this work, we deliberately did not elaborate a few points as they would have gone beyond the scope. However, some of these points are particularly interesting and should be examined in more detail in future work.

One of these points is dispute resolution in the event that providers and consumers cannot agree. In our work, disputes are resolved by the broker which monitors the data transfer. Because the broker has the knowledge of how much data was actually transferred, he can use the settlement contract in the event of a dispute to resolve the dispute by disclosing his knowledge. The disadvantage of this solution is that both providers and consumers have to trust the broker. It is entirely up to the broker to decide how much data has been transferred. There is of course great potential for misconduct. The broker could potentially manipulate the counter just to harm the entire system or work with another party to cheat. Future work could deal with how dispute resolution can be implemented without the broker having to be trusted.

Another problem, that is extremely important is how products can be discovered. We have implemented a discovery mechanism for products, but there is great potential for improvement. With our solution, we send a search query to the broker, which then iterates over all products and filters the products which satisfy the search query. However, the search results can only be narrowed down roughly, since only little metadata is saved with a product. We only use a simple string which describes the data type of the product. Certain information can be encoded in this string, but this does not offer a satisfactory solution. For future work it might be interesting to consider how the products of the data marketplace can be described with the help of metadata and how this metadata can be stored.

The rating system that we have implemented also offers opportunities for improvement. Since the support of fixed point numbers for Solidity has not yet been implemented, we use a simple counter as a rating, which is slightly increased in the case of positive behavior (no dispute) and greatly reduced in the case of negative behavior (dispute). Here, even more factors could be considered that can influence the rating and more complex calculation methods can be implemented to optimize the rating system. Thus, participants who behave incorrectly could be filtered out more effectively, so that a certain quality standard can be guaranteed. Honest participants should be able to use the data marketplace without constantly having problems with malicious participants.

Negotiations are also an essential point, which can be examined in future work. We have implemented a very simple negotiating process. This has the advantage that everything could be implemented relatively easy via smart contracts and thus the entire negotiation process is recorded on the blockchain. The disadvantage here, however, is that the use of smart contracts also results in relatively high costs. Future work could look at how to make the negotiation process more detailed and how to optimize costs.

It would be interesting not only to optimize the costs of the negotiation process but also the costs of the entire data trade. In the evaluation, we were able to determine that a data trade without negotiation already costs several dollars. This makes data trading not worthwhile for smaller amounts of data that do not exceed a certain total value.

The above-mentioned possibilities for future work are certainly not all. There are of course other possibilities for improvements and extensions possible. Finally, we can state that the solution presented here covers all key elements of an IoT data marketplace, but surely leaves room for improvements.

List of Figures

List of Tables

Bibliography

[A+09] Kevin Ashton et al. That 'internet of things' thing. *RFID journal*, 22(7):97–114, 2009.

[ABB+18] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, page 30. ACM, 2018.

[AFGM+15] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE communications surveys & tutorials*, 17(4):2347–2376, 2015.

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[Ant14] Andreas M Antonopoulos. *Mastering Bitcoin: unlocking digital cryptocurrencies*. " O'Reilly Media, Inc.", 2014.

[ASSC02] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: a survey. *Computer networks*, 38(4):393–422, 2002.

[BDPVA13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, pages 313–314, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Ber09] Daniel J Bernstein. Introduction to post-quantum cryptography. In *Post-quantum cryptography*, pages 1–14. Springer, 2009.

[BG17] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[BR18] Prabal Banerjee and Sushmita Ruj. Blockchain enabled data marketplace-design and challenges. *arXiv preprint arXiv:1811.11462*, 2018.

[But13] Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform. `https://github.com/ethereum/wiki/wiki/White-Paper`, 2013. [Online; accessed 9-September-2019].

[But16] Vitalik Buterin. A proof of stake design philosophy. `https://medium.com/@VitalikButerin/a-proof-of-stake-design-philosophy-506585978d51`, 2016. [Online; accessed 16-September-2019].

[CD16] Konstantinos Christidis and Michael Devetsikiotis. Blockchains and smart contracts for the internet of things. *IEEE Access*, 4:2292–2303, 2016.

[CL+99] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[CMVM18] Anamika Chauhan, Om Prakash Malviya, Madhav Verma, and Tejinder Singh Mor. Blockchain and scalability. In *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 122–128. IEEE, 2018.

[CPV+16] Tien-Dung Cao, Tran-Vu Pham, Quang-Hieu Vu, Hong-Linh Truong, Duc-Hung Le, and Schahram Dustdar. MARSA: A Marketplace for Realtime Human Sensing Data. *ACM Transactions on Internet Technology*, 16(3):16:1–16:21, 2016.

[CSC+05] Yingwei Cui, Narayanan Shivakumar, Alexander Carobus, Deepak Jindal, and Steve Lawrence. Content-targeted advertising using collected user behavior data, January 27 2005. US Patent App. 10/649,585.

[CT19] IOTA Foundation Coordicide Team. The coordicide. *Working Paper*, 2019.

[Dan15] Quynh H Dang. Secure hash standard. Technical report, NIST, 2015.

[DFZ+15] Yucong Duan, Guohua Fu, Nianjun Zhou, Xiaobing Sun, Nanjangud C Narendra, and Bo Hu. Everything as a service (xaas) on the cloud: origins, current and future trends. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 621–628. IEEE, 2015.

[DKJ16] Ali Dorri, Salil S Kanhere, and Raja Jurdak. Blockchain in internet of things: challenges and solutions. *arXiv preprint arXiv:1608.05187*, 2016.

[DSPSNAHJ17] Sergi Delgado-Segura, Cristina Pérez-Solà, Guillermo Navarro-Arribas, and Jordi Herrera-Joancomartí. A fair protocol for data trading based on bitcoin transactions. *Future Generation Computer Systems*, 2017.

[DV18] Alex De Vries. Bitcoin's growing energy problem. *Joule*, 2(5):801–805, 2018.

[fab19] A blockchain platform for the enterprise: Hyperledger fabric. `https://hyperledger-fabric.readthedocs.io/en/release-1.4/index.html`, 2019. [Online; accessed 24-September-2019].

[FCFL18] Tiago M Fernández-Caramés and Paula Fraga-Lamas. A review on the use of blockchain for the internet of things. *IEEE Access*, 6:32979–33001, 2018.

[GKJ18] Pooja Gupta, Salil S. Kanhere, and Raja Jurdak. A decentralized iot data marketplace. In *3rd Symposium on Distributed Ledger Technology (SDLT)*, 2018.

[Har19] Cara Harbor. Part 1: Iota data marketplace - update. `https://blog.iota.org/part-1-iota-data-marketplace-update-5f6a8ce96d05/`, 2019. [Online; accessed 25-February-2019].

[HHL18] Yuichi Hanada, Luke Hsiao, and Philip Levis. Smart contracts for machine-to-machine communication: Possibilities and limitations. In *2018 IEEE International Conference on Internet of Things and Intelligence System (IOTAIS)*, pages 130–136. IEEE, 2018.

[IHS16] IHS. Internet of things (iot) connected devices installed base worldwide from 2015 to 2025 (in billions). `https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/`, 2016. [Online; accessed 04-October-2019].

[JCZ12] Marijn Janssen, Yannis Charalabidis, and Anneke Zuiderwijk. Benefits, Adoption Barriers and Myths of Open Data and Open Government. *Information Systems Management*, 29(4):258–268, 2012.

[JFFL12] Xiaolin Jia, Quanyuan Feng, Taihua Fan, and Quanshui Lei. Rfid technology and its applications in internet of things (iot). In *2012 2nd international conference on consumer electronics, communications and networks (CECNet)*, pages 1282–1285. IEEE, 2012.

[JMV01] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International Journal of Information Security*, 1(1):36–63, Aug 2001.

[KAEM13] Stephen Kaisler, Frank Armour, J. Alberto Espinosa, and William Money. Big data: Issues and challenges moving forward. In *2013 46th Hawaii International Conference on System Sciences*, pages 995–1004. IEEE, 2013.

[KPCK17] Bhaskar Krishnamachari, Jerry Power, Shahabi Cyrus, and Seon Ho Kim. Iot marketplace: A data and api market for iot devices. `https://msbfile03.usc.edu/digitalmeasures/gerardpo/intellcont/USCIoTMarketplace_Jan152017-1.pdf`, 2017. [Online; accessed 29-October-2019].

[Ksh17] Nir Kshetri. Can blockchain strengthen the internet of things? *IT professional*, 19(4):68–72, 2017.

[LDBA17] Thomas Lundqvist, Andreas De Blanche, and H Robert H Andersson. Thing-to-thing electricity micro payments using blockchain technology. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6. IEEE, 2017.

[LeM18] Colin LeMahieu. Nano: A feeless distributed cryptocurrency network. *URL: https://nano. org/en/whitepaper*, 2018.

[MAM+99] Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Carlisle Adams. X. 509 internet public key infrastructure online certificate status protocol-ocsp. Technical report, RFC 2560, 1999.

[MBC+17] Paolo Missier, Shaimaa Bajoudah, Angelo Capossele, Andrea Gaglione, and Michele Nati. Mind my value: a decentralized infrastructure for fair and trusted iot data trading. In *Proceedings of the Seventh International Conference on the Internet of Things*, page 15. ACM, 2017.

[MCP+02] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97. Acm, 2002.

[MŽ16] Krešimir Mišura and Mario Žagar. Data marketplace for internet of things. In *2016 International Conference on Smart Systems and Technologies (SST)*, pages 255–260. IEEE, 2016.

[N+08] Satoshi Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system. *Bitcoin Whitepaper*, 2008.

[NBF+16] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and cryptocurrency technologies: A comprehensive introduction*. Princeton University Press, 2016.

[neo14] Neo white paper. https://docs.neo.org/docs/en-us/index.html, 2014. [Online; accessed 20-September-2019].

[NYGEV19] Christopher Natoli, Jiangshan Yu, Vincent Gramoli, and Paulo Esteves-Verissimo. Deconstructing blockchains: A comprehensive survey on consensus, membership and structure. *arXiv preprint arXiv:1908.08316*, 2019.

[PD16] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016.

[Pea19] William Peaster. Ethereum "eth 2.0" genesis block may launch in january 2020. https://blockonomi.com/ethereum-eth-2-0-genesis-block-january-2020/, 2019. [Online; accessed 23-October-2019].

[Pop17] Serguei Popov. The tangle. *Iota Whitepaper v1.3*, 2017.

[PZCG14] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Sensing as a service model for smart cities supported by internet of things. *Transactions on Emerging Telecommunications Technologies*, 25(1):81–93, 2014.

[Ram16] David Ramel. Microsoft closing azure datamarket. https://adtmag.com/articles/2016/11/18/azure-datamarket-shutdown.aspx, 2016. [Online; accessed 25-October-2019].

[RK18] Rahul Radhakrishnan and Bhaskar Krishnamachari. Streaming data payment protocol (sdpp) for the internet of things. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1679–1684. IEEE, 2018.

[RRK18] Gowri Sankar Ramachandran, Rahul Radhakrishnan, and Bhaskar Krishnamachari. Towards a decentralized data marketplace for smart cities. In *2018 IEEE International Smart Cities Conference (ISC2)*, pages 1–8. IEEE, 2018.

[SBV18] Joao Sousa, Alysson Bessani, and Marko Vukolic. A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform. In *2018 48th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pages 51–58. IEEE, 2018.

[SSV13] Fabian Schomm, Florian Stahl, and Gottfried Vossen. Marketplaces for data: an initial survey. *ACM SIGMOD Record*, 42(1):15–26, 2013.

[Sub18] Hemang Subramanian. Decentralized blockchain-based electronic marketplaces. *Commun. ACM*, 61(1):78–84, 2018.

[SZ15] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.

[Sza94] Nick Szabo. Smart contracts. `http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html`, 1994. [Online; accessed 9-September-2019].

[Sza97] Nick Szabo. The idea of smart contracts. `https://nakamotoinstitute.org/the-idea-of-smart-contracts/`, 1997. [Online; accessed 9-September-2019].

[Sø17] David Sønstebø. Iota data marketplace. `https://blog.iota.org/iota-data-marketplace-cb6be463ac7f`, 2017. [Online; accessed 25-October-2019].

[WG18] Karl Wüst and Arthur Gervais. Do you need a blockchain? In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 45–54. IEEE, 2018.

[Woo14] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 2014.

[WvB14] Dominic Wörner and Thomas von Bomhard. When your sensor earns money: exchanging data for cash with bitcoin. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, pages 295–298. ACM, 2014.

[ZLM+18] Yanqi Zhao, Yannan Li, Qilin Mu, Bo Yang, and Yong Yu. Secure pub-sub: Blockchain-based fair payment with reputation for reliable cyber physical systems. *IEEE Access*, 6:12295–12303, 2018.

[ZPG12] Arkady Zaslavsky, Charith Perera, and Dimitrios Georgakopoulos. Sensing-as-a-service and big data. In *2012 International Conference on Advances in Cloud Computing (ACC)*, pages 21–29, Bangalore, July 2012.