Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Artur Iablokov

Kostenanalyse einer Blockchain-basierten IoT-Infrastruktur mit Ethereum und Swarm

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Artur Iablokov

Kostenanalyse einer Blockchain-basierten IoT-Infrastruktur mit Ethereum und Swarm

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Martin Becke
Zweitgutachter: Prof. Dr. Jan Sudeikat

Eingereicht am: 8. August 2019

Artur Iablokov

Thema der Arbeit

Kostenanalyse einer Blockchain-basierten IoT-Infrastruktur mit Ethereum und Swarm

Stichworte

IoT, Blockchain, Backend, Skalierbarkeit, Ethereum, Swarm

Kurzzusammenfassung

Das Internet der Dinge (IoT) ist die Schlüsseltechnologie um Anwendungen wie Smart-Home, Smart-City, Industrie 4.0 und vielen weiteren zu ermöglichen [34]. Nach verschiedenen Forschungsberichten wird die Anzahl der angeschlossenen IoT-Geräte bis 2020 voraussichtlich 20 bis 50 Milliarden erreichen [41]. Die ständig wachsende Anzahl von IoT-Geräten erfordert eine sichere und skalierbare Infrastruktur zur Speicherung und Verarbeitung der erzeugten Daten. Eine Möglichkeit, Vertrauenswürdigkeit in IoT-Transaktion zu schaffen, besteht in einem verteilten Dienst, dem alle Teilnehmer vertrauen und der garantiert, dass die Daten unveränderlich bleiben.

Blockchain ist mit seiner dezentralen, vertrauenswürdigen Architektur eine passende Wahl. Die Hauptmerkmale von Blockchain, wie Sicherheit, Vertrauen und Identität der Dinge [27], machen diese Technologie besonders attraktiv für IoT. Das wird auch durch Statistiken bestätigt: der Einsatz von Blockchain für das Internet der Dinge wurde im Jahr 2018 verdoppelt [14]. Mit der Blockchain-Technologie gibt es hierzu einen einheitlichen Weg für die Integration aller Arten von IoT-Geräten in eine gemeinsame Infrastruktur [22, 35]. Jedoch haben IoT-Anwendungen sehr spezifische Eigenschaften, sie erzeugen große Datenmengen und erfordern Konnektivität und Stromversorgung für lange Zeiträume. Dies, zusammen mit den Einschränkungen bei Speicher, Rechnerkapazität, Netzwerken und begrenzter Stromversorgung, stellt eine Vielzahl von Herausforderungen bei der Integration von Blockchain und IoT dar.

Diese Bachelorthesis befasst sich mit der Skalierbarkeitsanalyse des vorgeschlagenen Blockchain-basierten IoT-Backends. Während der Studie wurden die IoT-Geräte als Blockchain Knoten konfiguriert und in ein gemeinsames privates Ethereumnetzwerk eingebunden. In der

jeweiligen Architektur wurde ein Swarm Peer-to-Peer Netzwerk als Datenspeicher verwendet und ein Smart Contract wurde bereitgestellt, um die Interaktion mit der Blockchain zu ermöglichen. Das System wurde innerhalb von 36 Stunden einer linear steigenden Datenlast ausgesetzt.

Im Laufe des Experiments wurde das Testnetzwerk überwacht und die Systemkosten, in erster Linie Latenz, Fehlerrate und Speicher, protokolliert. Die Latenz des betrachteten Blockchain-basierten Backends hat eine lineare Steigerung gezeigt, was durch die Besonderheiten des verwendeten Smart Contract verursacht wurde. Bei einer durchschnittlichen Latenz von 26296 ms lag die Fehlerrate bei 1,94%. Für die Ethereumgröße und Swarmgröße konnte ebenfalls ein lineares Wachstum im Zusammenhang mit einer anwachsenden Datenmenge festgestellt werden. Diese betrug für beide jeweils etwa 11 MB zum Ende des Experiments.

Artur Iablokov

Title of Thesis

Cost analysis of a blockchain-based IoT infrastructure with Ethereum and Swarm

Keywords

IoT, Blockchain, Backend, Scalability, Ethereum, Swarm

Abstract

The Internet of Things (IoT) is the key technology to enable applications like Smart-Home, Smart-City and Industry 4.0 [34]. According to various research papers, the number of connected IoT devices is expected to reach 20 to 50 billion by 2020 [41]. The constantly growing number of IoT devices requires a secure and scalable infrastructure for storing and processing the generated data. One way to achieve reliability in IoT transactions is to have a distributed service that all participants trust and which guarantees that the data remains immutable.

Blockchain with its decentralized, trustless architecture is a suitable choice. The main features of Blockchain, such as security, trust and identity of things [27], make this technology particularly attractive for IoT. The usage of blockchain for the Internet of

Things has been doubled in 2018 [14]. Blockchain technology provides a clear path for integrating all kinds of IoT devices to a common blockchain-based infrastructure [22, 35]. However, IoT applications have very specific characteristics: they generate large amounts of data and require connectivity and power supply for a long time. This, along with the limitations of memory, computing capacity, networks and limited power supply, presents a variety of challenges to integration of blockchain and IoT.

This bachelor thesis focuses on the scalability analysis of the proposed blockchain-based IoT backend. During the study, the IoT devices were configured as blockchain nodes and integrated into a common private Ethereum network. In the respective architectures, a Swarm Peer-to-Peer network was used as the data storage and Smart Contract was provided to enable interaction with the blockchain. The system was exposed to a linearly increasing data load within 36 hours.

Throughout the experiment, the test network was monitored and system costs, primarily latency, error rate and memory, were logged. The latency of the considered blockchain-based backend showed a linear increase, which was caused by the special characteristics of the Smart Contract used. With an average latency of 26296 ms, the error rate was 1,94%. For the size of Ethereum and Swarm, linear growth was also observed with an increasing amount of data. This was about 11 MB for both networks at the end of the experiment.

Inhaltsverzeichnis

Abbildungsverzeichnis **viii**

Tabellenverzeichnis **ix**

1 Einleitung **1**

1.1 Motivation 1
1.2 Zielsetzung 2
1.3 Aufbau der Arbeit 3

2 Grundlagen **4**

2.1 Blockchain 4
2.1.1 Entwicklung der Blockchain 4
2.1.2 Funktionsweise der Blockchain 5
2.1.3 Diskussion 6
2.1.4 Blockchain Plattformen 7
2.2 Ethereum 8
2.2.1 Ethereum Plattform 9
2.2.2 Konten 11
2.2.3 Globaler Systemzustand 12
2.2.4 Mining 14
2.2.5 Vergütung 14
2.2.6 Transaktionen 15
2.2.7 Smart Contract 16
2.2.8 Geth 16
2.2.9 Swarm 17
2.3 IoT und Blockchain 18
2.3.1 Blockchain-IoT Interaktionen 19
2.3.2 Herausforderungen bei der Blockchain-IoT Integration 22

3 Experiment **26**
3.1 Testnetzwerk . . . 26
3.1.1 Backend Design . . . 26
3.1.2 Testnetzwerk Design . . . 29
3.1.3 Testtool . . . 31
3.2 Testnetzwerk Setup . . . 33
3.2.1 Vorbedingungen . . . 33
3.2.2 Ethereum . . . 35
3.2.3 Swarm . . . 36
3.2.4 Smart Contract . . . 36
3.2.5 Testtool . . . 37
3.3 Experimentablauf . . . 38

4 Ergebnisse **42**
4.1 Latenz . . . 42
4.2 Fehlerrate . . . 49
4.3 Speicher . . . 50

5 Schlussfolgerungen **53**

Literaturverzeichnis **55**

A Appendix **60**
A.1 genesis.json . . . 60
A.2 default.json . . . 61
A.3 vendor.sol . . . 62

Selbstständigkeitserklärung **66**

Abbildungsverzeichnis

2.1 Der Aufbau der Blockchain [16] 5
2.2 Ethereum Zustandsautomat [20] 9
2.3 Ethereum Pfaden - Verzweigung [20] 11
2.4 Merkle-Baum [20] 13
2.5 Blockchain-IoT Interaktion 20

3.1 IoT-Backend Architektur 27
3.2 Datenspeicherung und Datenerfassung 29
3.3 Testnetzwerkplan 30
3.4 Testtool: Komponentendiagramm 32

4.1 node0: Latenz 43
4.2 node1: Latenz 43
4.3 node2: Latenz 43
4.4 node0 - Latenz: Abweichung von Mittelwert 44
4.5 node1 - Latenz: Abweichung von Mittelwert 44
4.6 node2 - Latenz: Abweichung von Mittelwert 44
4.7 Latenz: Mittelwert pro Gerät 46
4.8 Latenz: Mittelwert 46
4.9 Latenz: Mining 46
4.10 node0 - Latenz: Transaktionsbestandteile 47
4.11 node1 - Latenz: Transaktionsbestandteile 47
4.12 node2 - Latenz: Transaktionsbestandteile 47
4.13 Latenz: Stückkosten 48
4.14 Fehlerrate 50
4.15 Ethereumgröße 51
4.16 Swarmgröße 51

Tabellenverzeichnis

2.1 Blockchain-Plattformen [39] 7

3.1 Beschreibung der Geräte 30
3.2 Experiment: Datenlast pro Stunde 39

4.1 Latenz Übersicht 42
4.2 Fehlerrate Übersicht 49

1 Einleitung

1.1 Motivation

Mit dem Aufkommen von Smart Homes, Smart Cities und Smart Everything hat sich das Internet der Dinge (IoT) zu einem Bereich mit unglaublicher Wirkung, Potenzial und Wachstum entwickelt. Cisco Inc. prognostiziert, dass im Jahr 2020 50 Milliarden angeschlossene Geräte verfügbar sein werden [40].

IoT-Geräte tauschen in erster Linie Sensor- und Kontrolldaten in Echtzeit in kleinen, aber zahlreichen Nachrichten aus. Auf diese Art und Weise erzeugen IoT-Geräte eine enorme Menge an Daten, die dennoch sicher gespeichert werden müssen. Die meisten dieser IoT-Geräte haben jedoch viele Sicherheitsprobleme und können mühelos kompromittiert werden. [32]

Die ständig wachsende Anzahl von IoT-Geräten erfordert eine sichere und skalierbare Infrastruktur zur Speicherung und Verarbeitung der erzeugten Daten. Die Suche nach der optimalen Architektur für IoT-Plattformen und -Dienste ist eine offene wissenschaftliche Frage [7, 23]. Bereits bestehende Lösungen stehen vor zahlreichen Herausforderungen wie die der Standardisierung, der Heterogenität, der Sicherheit und der IoT-Geräte-Identifikation [38].

Die Blockchain-Technologie ist einer der Kandidaten, um die Spielregeln in der IoT-Welt zu ändern. Die Erwartungen der IT-Community an die Integration der beiden Technologien sind hoch, was ebenfalls durch das hohe wissenschaftliche Interesse bestätigt wird [18, 44]. Die Blockchain mit ihrer dezentralen, vertrauenswürdigen Architektur bietet einen einheitlichen Weg für die Integration aller Arten von IoT-Geräten in eine gemeinsame Infrastruktur [22, 35].

Natürlich löst die Blockchain nicht alle Probleme auf einmal, sondern fügt teilweise neue hinzu, die mit der Natur der Blockchain selbst zusammenhängen. Während auf der einen

Seite die Blockchain einen hohen Ressourcenbedarf aufweist und mit Skalierbarkeitsproblemen zu kämpfen hat, zeichnen sich IoT-Geräte bisher durch Einschränkungen bezüglich ihrer Speicher- und Rechenkapazität sowie durch ihre Netzwerke und begrenzte Stromversorgung aus. Dies ergibt eine Vielzahl von Herausforderungen bei der Integration von Blockchain und IoT.

Die Skalierbarkeit erweist sich als Hauptstolperstein bei der Integration beider Technologien. Der Anspruch von IoT-Plattformen, in Echtzeit zu arbeiten, ist nur schwer mit den Mechanismen der Blockchain-Technologie zu kombinieren. Blockchain-Mechanismen, wie beispielsweise Synchronisation und Mining, führen zu einer Erhöhung der Latenz und einer Reduktion des Durchsatzes. Eine IoT-Plattform mit hoher Latenz und niedrigem Durchsatz ist jedoch zum Scheitern verurteilt, da dies dem Anspruch auf Verarbeitung der Daten in Echtzeit widerspricht.

Eine Studie von Özyılmaz und Yurdakul (2018) mit dem Namen „Designing a blockchain-based IoT infrastructure with Ethereum, Swarm and LoRa“ stellt die Architektur des Blockchain-basierten Backends dar [30]. In der betrachteten Architektur wurden IoT-Geräte verwendet, um gesammelte Daten per HTTP an ein Peer-to-Peer Swarm-Netzwerk zu übertragen. Die Ethereum Blockchain wurde als Datei-Explorer verwendet. Um diesen Prozess zu automatisieren, schlugen die Autoren einen Smart Contract vor.

Die von den Autoren vorgeschlagene Lösung basiert auf einem Ethereum-Technologie-Stack, der häufig genutzt wird in der Entwicklung von IoT-Plattformen [39]. Somit ist das oben genannte Backend-Projekt ein perfektes Vorbild für die Untersuchung der Skalierbarkeit Blockchain-basierter Systeme, wie sie im Rahmen dieser Arbeit durchgeführt wird.

1.2 Zielsetzung

Das Ziel dieser Bachelorarbeit ist es, Blockchain-basiertes IoT-Backend zu untersuchen. Betrachtet wird wie die Systemkosten in Form von Latenz, Fehlerrate und beanspruchtem Speicher gegen eine steigende Problemgröße bzw. die Anzahl von Transaktionen skalieren. Um dieses Ziel zu erreichen, wurden 2 IoT-Geräte und ein Server zu einer gemeinsamen Infrastruktur mit Ethereum und Swarm zusammengefasst. Das entwickelte Testtool erzeugte innerhalb von 36 Stunden eine linear wachsende Datenlast und protokollierte die

oben genannten System-Parameter. Als letztes wurden die Ergebnisse präsentiert und analysiert.

1.3 Aufbau der Arbeit

Die Bachelorarbeit ist wie folgt aufgebaut:

Kapitel 2 - Grundlagen - fasst für die vorliegende Arbeit relevante Grundlagenkenntnisse zusammen. Das Unterkapitel 2.1 dient als Hinführung zum Verständnis der Blockchain Technologie. In 2.2 wird auf die Blockchain-Plattform Ethereum und die Speicherplattform Swarm eingegangen. Dem schließt sich Unterkapitel 2.3 an, welches sich mit der Integration der IoT-Blockchain und damit verbundenen Problemen beschäftigt.

Kapitel 3 - Experiment - beschreibt den Aufbau und die Struktur des Experiments. Das Unterkapitel 3.1 beschäftigt sich mit dem Design des Blockchain-basierten Backends und dem Testtool. Es folgt das Unterkapitel 3.2, dessen Gegenstand es ist, eine Beschreibung der Konfiguration und Bereitstellung des Backends, sowie anderer Elemente des Experiments zu liefern. Zuletzt fügt sich Unterkapitel 3.3 an, in welchem der Ablauf des Experiments erläutert wird.

Kapitel 4 - Ergebnisse - stellt die Ergebnisse des Experiments zusammen. Der Abschnitt 4.1 konzentriert sich auf das Thema Latenz. Im Kapitel 4.2 wird die Error-Rate des betrachteten Backends bestimmt und die Ursachen der Fehler analysiert. Als letztes in 4.3 wird die Größe der Ethereum Blockchain und des Swarm betrachtet.

Kapitel 5 - Schlussfolgerungen - fasst abschließend die Forschungsergebnisse zusammen.

2 Grundlagen

In diesem Kapitel werden für die vorliegende Arbeit relevante Grundlagenerkenntnisse zusammengefaßt. Gegenstand von Unterkapitel 2.1 ist die Hinführung zur Blockchain Technologie. In Abschnitt 2.2 wird auf die Blockchain-Plattform Ethereum und ihre Elemente, insbesondere ihre Funktionsweisen, Smart-Contracts und die Speicherplattform Swarm eingegangen. Es schließt sich Unterkapitel 2.3 an, welches sich mit der IoT-Blockchain Integration und damit verbundenen Problemen beschäftigt.

2.1 Blockchain

Die Blockchain ist eine sogenannte „Distributed-Ledger"-Technologie [19]. Wörtlich lässt sich der Begriff als „verteiltes Kontobuch" übersetzen, im Grunde beschreibt sie eine öffentliche, dezentral geführte Datenbank (bzw. Register), welche über ein Netzwerk von verschiedenen Teilnehmern genutzt wird. In jenem Netzwerk verfügen Teilnehmer über ihre identische Kopie des Ledgers [46]. Die Netzwerke, in denen jeder Teilnehmer über den gesamten Datenbestand verfügen, werden als Peer-to-Peer-Netzwerk bezeichnet [19]. Im einem Peer-to-Peer-Netzwerke haben alle Teilnehmer (oft als Knoten oder „nodes" bezeichnet) dieselben Rechte und können auf die gleichen Informationen zugreifen, sowie dem Ledger Informationen zuführen. Ein einzelner Teilnehmer kann nicht darüber entscheiden, ob und welche Einträge dem Ledger eintreffen werden, stattdessen kommt dazu ein Konsensprozess zum Einsatz, in dem die Mehrheit der Teilnehmer den Eintrag verifiziert.

2.1.1 Entwicklung der Blockchain

Die Entstehung der Blockchain hängt eng mit der Entwicklung der Kryptowährung Bitcoin zusammen. Im November 2008 publizierte Satoshi Nakamoto ein technisches Konzept für digitales Bargeld unter dem Titel „Bitcoin: A Peer-to-Peer Electronic Cash System"

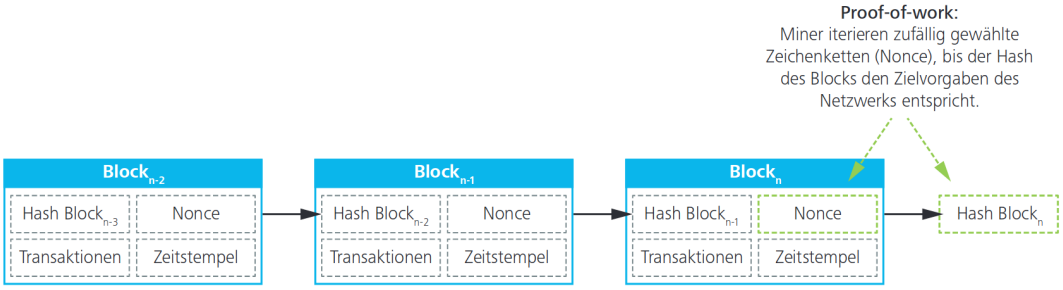

Abbildung 2.1: Der Aufbau der Blockchain [16]

[28]. Zu Beginn nutzten vor allem Kryptographie-Experten Bitcoins als Währung. Nakamoto war keineswegs der Erste, der sich an der Erschaffung digitalen Geldes versucht hat, doch waren die Versuche seiner Vorgänger daran gescheitert, dass eine zentrale Autorität alle Transaktionen im Währungsraum überwachen musste, um die Gefahr des Double Spending zu unterbinden. Nakamoto fand einen neuen Weg, das Double Spending zu verhindern: die Blockchain. [16]

Im Gegensatz zur Welt der körperlichen Gegenstände können digitale Güter unbegrenzt vervielfältigt werden. Dieses Phänomen wird als „Double Spending“ bezeichnet [6]. Dadurch ist es möglich sie zu nutzen, ohne dabei die Nutzungsmöglichkeiten anderer einzuschränken. Ein funktionierendes Währungssystem ist darauf angewiesen, dass seinen internen Werteinheiten exklusive Nutzungsrechte zugeordnet werden können [16].

2.1.2 Funktionsweise der Blockchain

Die Blockchain selbst ist eine Datenbank, in welcher alle Transaktionen verzeichnet werden. Im Rahmen des Validierungsverfahrens für Transaktionen („Mining”) werden Transaktionen in Blöcken zusammengefasst [16]. Jeder Block verweist auf den ihm vorangegangenen gültigen Block, wodurch eine Kette, die „block chain” entsteht (Abb. 2.1). Einige Netzwerkteilnehmer, die sogenannten „Miner”, wetteifern darum, neue Transaktionsblöcke zu erstellen, denn der erste Miner, der einen neuen Block findet, erhält eine Belohnung in Form neuer Tokens. Um einen Block zu finden, muss eine Berechnung erbracht werden oder eine andere Aktion je nach Konsens-Mechanismus ausgeführt werden.

Um einzelne Transaktionen in der Blockchain nachträglich zu verändern, muss der Konsens-Mechanismus für die nachfolgenden Blöcke neu erbracht werden. Zusätzlich müssen die

restlichen Teilnehmer die nachträglich geänderte Blockchain akzeptieren [36].

Im Gegensatz zu herkömmlichen Datenbanken läuft die Blockchain nicht auf einem einzelnen Server. Alle Teilnehmer des Netzwerks besitzen eine vollständige Kopie der kompletten Blockchain in ihrem lokalen Speicher. Man spricht deshalb von einer replizierten, geteilten Datenbank [16].

2.1.3 Diskussion

Die dargestellte Funktionsweise der Blockchain bleibt im Kern allerdings immer dieselbe: Daten und Informationen werden verschlüsselt, zu Blöcken zusammengefasst, validiert und dem Ledger hinzugefügt [46]. Dabei verweist jeder Datenblock auf den ihm vorangehenden Block, wodurch eine Blockkette entsteht. Weitere wesentliche Merkmale, die eine Blockchain i.d.R. aufweist sind:

- Digitale Verteilung der Blockchain auf mehreren Rechnern: In einem dezentralen System werden digitale Kopien der gesamten Blockchain auf den Rechnern der Netzwerkteilnehmer abgespeichert.
- Konsens-Modelle: Die Blockchain nutzt viele Teilnehmer in Netzwerk, um die Integrität der Daten zu validieren und einen Konsens darüber zu erreichen, welche Daten der Kette hinzugefügt werden [46].
- Blockchains nutzen Kryptografie und digitale Signaturen, um Identitäten zu bestimmen, Transaktionen nachzuverfolgen und diese gleichzeitig sicher und nicht für jeden einsehbar zu gestalten. Blockchains verfügen über Mechanismen, um es Angreifern möglichst schwierig zu machen, bestehende Daten zu manipulieren (z.B. Proof-of-Work-Konzepte).
- Blockchains sind programmierbar: In den Blöcken können Anweisungen eingebettet sein, die zu entsprechenden Aktionen führen, falls bestimmte Kriterien erfüllt werden, sowie weiterführende Daten zu Transkationen o.ä. enthalten.

Als Nachteile sind zu nennen:

- Mit jeder Transaktion wächst die Blockchain. Wenn nun jeder Teilnehmer des Netzwerkes die gesamte Blockchain auf seinem Rechner speichert, können sich Grenzen im Hinblick auf die Speicherkapazitäten ergeben. Der Erfolg einer Blockchain (mehr

Teilnehmer bedeuten mehr Transaktionen) stellt damit zugleich ein Risiko dar: Die Skalierbarkeit der Technologie steht infrage.

- Als ein nächstes Problem der Skalierbarkeit ergibt sich weiterhin, dass mit der derzeitigen Blockchain-Infrastruktur oft nur wenigen Transaktionen pro Sekunde durchführbar sind. Viele Branchen benötigen die Verarbeitung von Tausenden / Zehntausenden von Transaktionen pro Sekunde, sodass die Performance der Blockchain hierfür noch als unzureichend gelten muss.
- Die Blockchain Technologie ermöglicht die Konsensfindung in einem dezentralen Netzwerk. Die Art der Lösung ist dabei jedoch alles andere als effizient. Nach aktuellen Schätzungen benötigt eine Bitcoin-Transaktion beispielsweise mehr Strom als ein Haushalt an einem Tag. [46]

2.1.4 Blockchain Plattformen

Die Blockchain wurde bereits in vielen Bereichen eingesetzt, darunter Kryptowährungen, Zahlungssysteme, in der Versicherungsbranche und Immobilienbranche, in der Energiebranche, im E-Government und viele anderen Bereichen [39]. Bestehende Blockchain-Plattformen werden typischerweise genutzt, um Blockchain-basierte Anwendungen in all diesen Branchen zu implementieren. Die Anzahl der Plattformen ist so hoch und einige von diesen befinden sich gleichzeitig in solch stetigem Wandel, dass es unmöglich ist, sie alle zu analysieren. Eine Studie von Reyna und Martin (2018) mit dem Namen „On blockchain and its integration with IoT. Challenges and opportunities“ untersucht die beliebtesten und am besten geeigneten für IoT Blockchain-Plattformen. Diese werden in der Tabelle 2.1 dargestellt.

Tabelle 2.1: Blockchain-Plattformen [39]

Plattform	Blockchain	Konsens-Mechanismus	Kryptowährung	Smart Contract
Ethereum	Public, permission-based	PoW	ja	ja
Hyperledger Fabric	Permission-based	PBTF/SIEVE	nein	ja
Multichain	Permission-based	PBTF	ja	ja
Litecoin	Public	Scrypt	ja	nein
Quorum	Permission-based	CFT/BFT	ja	ja

Hyperledger [1] ist eine Open-Source-Plattform, auf der verschiedene Projekte im Zusammenhang mit Blockchain entwickelt wurden, darunter Hyperledger Fabric, eine Block-

chain ohne Berechtigungen und ohne Kryptowährung, auf der kommerzielle Implementierungen wie IBMs Blockchain-Plattform basieren. Hyperledger bietet verschiedene Komponenten für Konsens. Smart Contracts können in der Hyperledger-Blockchain mit Hilfe von üblichen Programmiersprachen entwickelt werden. IoT-Geräte können über die IBM Watson IoT-Plattform Daten an die Blockchain liefern.

Die Multichain-Plattform [3] ermöglicht die Erstellung und Bereitstellung von privaten Blockchains. Multichain verwendet eine API, die den Kern der ursprünglichen Bitcoin-API um neue Funktionen erweitert und die Verwaltung von Berechtigungen, Transaktionen usw. ermöglicht.

Litecoin [2] ist technisch identisch mit Bitcoin, zeichnet sich aber durch schnellere Transaktionsbestätigungszeiten und eine verbesserte Speichereffizienz durch die Reduzierung der Blockgenerierungszeit aus. Dies bedeutet, dass der Rechenbedarf der Litecoin-Knoten geringer ist, so dass sie für das IoT besser geeignet sind.

Quorum [4] ist eine Blockchain-Plattform, welche für die Finanzdienstleistungsbranche mit dem Schwerpunkt auf Datenschutz entwickelt wurde. Sie ermöglicht mehrere Konsensmechanismen und erreicht den Datenschutz durch Kryptographie und Segmentierung. Die Plattform hat kürzlich die ZeroCash-Technologie integriert, um alle identifizierbaren Informationen über eine Transaktion zu verschleiern.

Ethereum ist die Plattform, die seit seiner Gründung im Jahr 2015 eine wichtige Rolle spielt. Ethereum gehörte zu den Pionieren bei der Umsetzung von Smart Contract. Die Umsetzung von Smart Contract schiebt die Blockchain fort von den Kryptowährungen und erleichtert die Integration dieser Technologie in neue Bereiche. Jene Aspekte kombiniert mit seiner aktiven und breiten Community hat Ethereum zum Rang der beliebtesten Plattform der Blockchainentwickler verholfen. Die meisten IoT-Anwendungen verwenden Ethereum oder sind damit kompatibel [39]. Im Folgenden wird die Ethereum-Plattform näher erläutert.

2.2 Ethereum

In diesem Unterkapitel geht es um die Ethereum-Plattform. Ziel dieses Abschnitts ist es, eine erste Vorstellung davon zu vermitteln, wie Ethereum funktioniert, aus welchen Elementen es besteht und die dahinterstehenden Konzepte für diese Arbeit zu erklären.

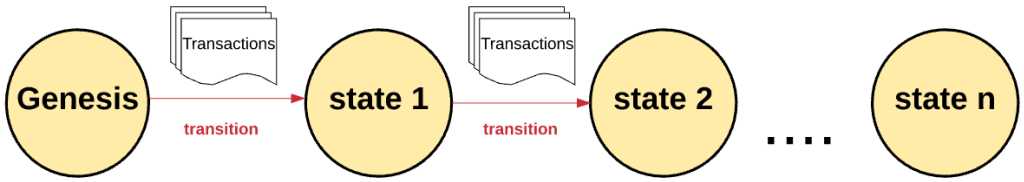

Abbildung 2.2: Ethereum Zustandsautomat [20]

Einige Ethereum-Konzepte werden nur oberflächlich betrachtet (Transaktionen, Mining) oder ganz weggelassen (EVM, Execution Model).

Das Ethereum ist eine komplexe Plattform, an der Tausende von Menschen auf der ganzen Welt arbeiten. Dieses Kapitel stellt nur eine kurze Zusammenfassung der ursprünglichen Ethereum Yellow Paper [51] dar und fokussiert an wichtigsten Aspekten für diese Arbeit.

2.2.1 Ethereum Plattform

Ethereum Blockchain ist ein Transaktionsstatus-System und ähnelt teilweise einem Zustandsautomaten. Ein Zustandsautomat ist ein System, das Eingangsinformationen verarbeitet und diese in einen neuen Zustand überführt.

Im Ethereum-Zustandsautomat (Abb. 2.2) beginnen alle Prozesse mit dem Anfangszustand. Dieser Zustand entspricht einem Nullzustand, in dem sich der Automat befindet, bevor transaktionsbezogene Aktionen in seinem Netzwerk gestartet werden. Wenn solche Transaktionen ausgeführt werden, wird der Anfangszustand durch den Endzustand ersetzt, wobei der Endzustand jederzeit den aktuellen Zustand des Ethereums anzeigt.

Die Ethereum Transaktionen werden zu Blöcken zusammengefasst. Ein Block enthält eine Reihe von Transaktionen und jeder nachfolgende Block ist mit dem vorherigen verbunden, wodurch eine eindeutige Kette von Blöcken (Blockchain) entsteht.

Die Transaktion muss korrekt sein, damit sie von einem Zustand zum anderen wechselt. Eine Transaktion gilt erst dann als korrekt, wenn sie den Verifizierungsprozess - das sogenannte *Mining* - bestanden hat. Beim *Mining* stellt eine Gruppe von Knoten (Computer, Server, IoT-Geräte) ihre Rechenressourcen zur Verfügung, um einen Block mit korrekten Transaktionen zu erstellen.

Jeder Knoten im Netzwerk, der sich als *Miner* bezeichnet, ist in der Lage, einen Block von Transaktionen zu erstellen und zu verifizieren. Es ist üblich, dass mehrere *Miner* gleichzeitig versuchen, einen Transaktionsblock zu erstellen und zu verifizieren, wobei am Ende nur ein *Miner* mit seiner Verifizierung erfolgreich ist. Jeder *Miner* führt dabei seine eigene mathematische Berechnung durch, deren Ziel es ist, einen Beweis zu finden: wenn ein Beweis existiert, werden die Transaktionen im Block als korrekt angesehen.

Ein *Miner* muss seinen mathematischen Beweis schneller erbringen als jeder andere Wettbewerber, damit sein Block in der Blockchain hinzugefügt werden kann. Der Prozess der Überprüfung jedes Blocks, bei dem ein *Miner* seinen mathematischen Beweis liefern soll, wird als *Proof of Work* bezeichnet.

Ein *Miner*, der einen neuen Block erschafft, erhält eine gewisse Vergütung für die Ausführung der Arbeit. Die Ethereum Blockchain verwendet einen eingebauten digitalen Token namens *Ether*. Jedes Mal, wenn ein *Miner* seinen Transaktionsblock erfolgreich erzeugt, entsteht ein neuen Token beziehungsweise ein neuer *Ether* und der *Miner* wird für dessen Erschaffung belohnt.

Ethereum ist ein transaktionales Einzelelementsystem mit einem globalen Zustand. Aus dieser Definition kann man schließen, dass es nicht mehrere korrekten aktuellen Zustände gibt - es ist der einzige seiner Art. So muss jeder Teilnehmer des Systems diese Aussage als Wahrheit akzeptieren. Das Vorhandensein mehrerer Zustände würde das gesamte System zerstören, da es unmöglich wäre, sich zu einigen, welcher Zustand korrekt ist. Immer wenn mehrere Pfade erzeugt werden, kommt es zu einer „Verzweigung“ (Abb. 2.3). Eine Verzweigung ist stets unerwünscht, da sie die Integrität des Systems stört, und um dies zu verhindern, müssen die Teilnehmer eine der möglichen Ketten wählen.

Um festzustellen, welcher der möglichen Pfade korrekt ist, wird im Etherium GHOST-Protokoll (Greedy Heaviest Observed Subtree) verwendet. Das GHOST-Protokoll bestimmt, dass nur der Pfad mit der höchsten Anzahl von Beweisen ausgewählt werden soll. Für die Definition einer solchen Vorgehensweise wird die Nummer des zuletzt definierten Blocks verwendet. Dank dieses Ansatzes wird die Gesamtzahl der Blöcke im aktuellen Pfad bestimmt. Je höher der Block ist, desto länger ist der Pfad und desto mehr Beweise müssen die Miner bieten. Aus diesen Gründen wird nur die für den aktuellen Zustand korrekte Version akzeptiert.

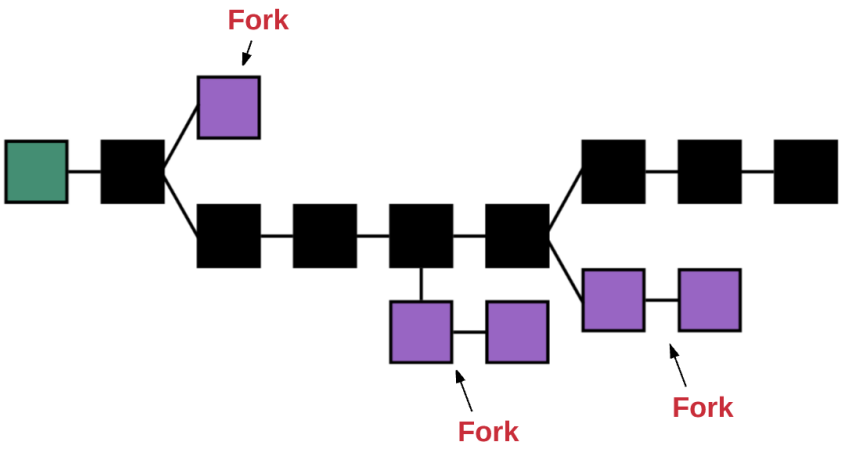

Abbildung 2.3: Ethereum Pfaden - Verzweigung [20]

2.2.2 Konten

Der globale Ethereum-Zustand besteht aus zahlreichen Konten, die durch den Austausch von Nachrichten miteinander interagieren. Jedes Konto hat einen bestimmten Status und eine 20-Byte-Adresse zur Identifizierung.

Es gibt zwei Arten von Konten:

- Externe Konten werden über private Schlüssel gesteuert.
- Smart Contract Konten werden durch einen speziellen Code gesteuert, der im Smart Contract festgelegt ist.

Ein externes Konto kann Nachrichten an andere externe Konten sowie an andere Smart Contract Konten senden. Zu diesem Zweck ist es notwendig, eine neue Transaktion mit dem privaten Schlüssel anzulegen. Die Nachricht zwischen zwei externen Konten ist nur ein zu übertragender Wert. Und andererseits bedeutet eine Nachricht, die von einem externen Konto an ein Smart Contract-Konto gesendet wird, die Aktivierung des Smart Contract Codes.

Im Gegensatz zu externen Konten ist es nicht möglich mit Hilfe von Smart Contract Konten selbstständig neue Transaktionen anzustoßen. Stattdessen können Smart Contract-Konten nur zur Ausführung von Transaktionen als Reaktion auf andere empfangene

Transaktionen verwendet werden. Jede Aktion in der Ethereum-Blockchain wird mit Hilfe von Transaktionen durchgeführt, die von externen Konten initiiert werden.

Der Kontostand, unabhängig von seiner Art, wird durch 4 Werte beschrieben:

- nonce: Wenn es sich um ein externes Konto handelt, dann beschreibt nonce die Anzahl der Transaktionen, die von diesem Konto gesendet wurden. Wenn das Konto ein Smart Contract Konto ist, stellt *nonce* die Anzahl der auf dem Konto angelegten Verträge dar.
- balance: Die Tokensumme, die zu diesem Konto gehören, wird als *balance* bezeichnet. Die Ethereum-Austauscheinheit heißt *wei* (10^{18} *wei* = 1 *ether*).
- storageRoot: *StorageRoot* ist der Hash der Wurzel des Merkle-Baum.
- codeHash: Der Hash des EVM-Codes des Kontos. Bei Smart Contract-Konten ist dieses Feld ein Hash-Wert des Codes.

2.2.3 Globaler Systemzustand

Der globale Ethereum-Zustand ist also die Zuordnung der Kontoadressen zu den Kontoständen. Dieser Vergleich wird in der Datenstruktur – dem Merkle-Baum – gespeichert. Der Merkle-Baum (Abb. 2.4) ist eine Binärdatei, die aus einer Reihe von Knoten besteht:

- normale Knoten („Baumblätter“), die sich am unteren Teil des Baums befinden, der die Grunddaten enthält,
- Zwischenknoten, wobei jeder Knoten ein Hash seiner zwei Tochterknoten ist,
- ein Wurzelknoten, der die Spitze des Baumes darstellt.

Eine entsprechende Baumstruktur wird für die Speicherung von Daten und Transaktionen verwendet.

Der Hash im Merkle-Baum breitet sich von den unteren Zweigen auf die oberen Zweige aus, und wenn ein Teilnehmer versucht, die ursprüngliche Transaktion durch eine gefälschte im unteren Teil des Merkle-Baums zu ersetzen, ändert er den Hash des oberen Knotens, der wiederum den Hash des darüber liegenden Knotens ändert, und so weiter, bis er schließlich die Wurzel ändert.

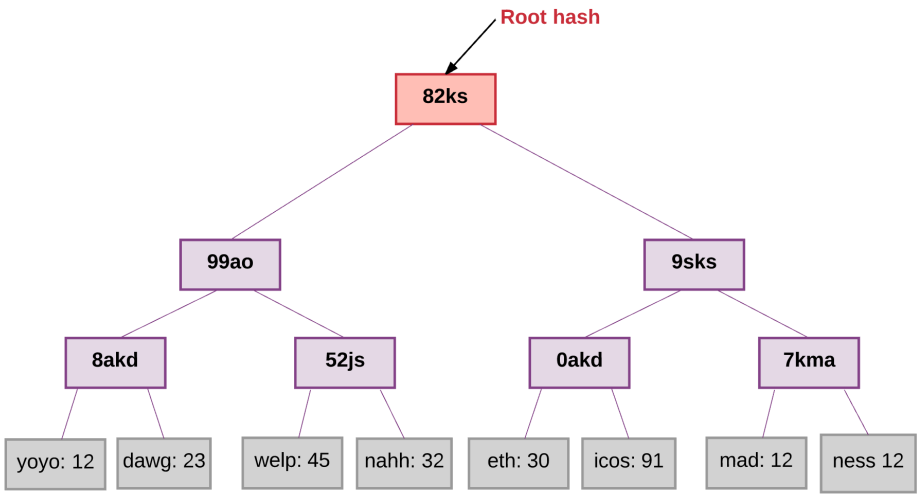

Abbildung 2.4: Merkle-Baum [20]

Der Wurzelknoten des Merkle-Baums ist abhängig von den im Baum gespeicherten Daten. Daher kann man mit Hilfe des Hash des Wurzelknotens diese Daten als sicher identifizieren. Da der Block-Header den Wurzel-Hash der Bäume enthält, können sämtliche Knoten die unterschiedlichen Teile des Ethereum-Zustands überprüfen, ohne alle Zustände speichern zu müssen.

Die Blockchain funktioniert mit einer Reihe von Knoten. Es gibt zwei Arten von Knoten im Etherium: Full und Light. Der Full-Knoten synchronisiert die gesamte Blockchain. Die gesamte Kette (Chain) wird vom Anfangszustand bis zum aktuellen Endzustand geladen und alle darin befindlichen Transaktionen werden ausgeführt. Miner sind Full-Knoten, sie benötigen ein vollständiges Archiv, um am Mining-Prozess teilzunehmen. Wenn ein Knoten nicht jede einzelne Transaktion ausführen oder die gesammelten Daten anfordern muss, kann die Speicherung der gesamten Kette redundant sein. In diesem Fall ist die Verwendung eines Light-Knotens relevant. Anstatt die gesamte Kette zu laden und alle Transaktionen auszuführen, laden Light-Knoten nur die Kette der Header-Dateien, ohne Transaktionen auszuführen. Da Light-Knoten Zugriff auf Block-Header haben, können sie problemlos relevante Daten zu Transaktionen, Ereignissen, Saldos und so weiter erstellen und abrufen.

2.2.4 Mining

Der im Ethereum-System verwendete *Proof of Work*-Algorithmus wird als *Ethash* bezeichnet. Dieser Algorithmus hat die folgende Form:

$$(m, n) = \mathsf{PoW}(H_m, H_n, \mathsf{d})$$

wobei m für mixHash, n für nonce, Hn für den Header des neuen Blocks, Hn für nonce des Block-Headers und d für den DAG-Datensatz steht.

MixHash und nonce sind Felder im Block-Header, die vom oben genannten Algorithmus berechnet werden. Im Allgemeinen besteht der Zweck von *Proof of Work (PoW)* darin, kryptographisch nachzuweisen, dass bestimmte Berechnungen darauf abzielten, ein bestimmtes Ergebnis (nonce-Wert) zu erhalten. Es stellt sich also heraus, dass es keinen anderen Weg gibt, nonce zu finden, als alle möglichen Optionen auszuprobieren. Die Wahrscheinlichkeit den richtigen Beweis der PoW-Berechnung zu finden, ist zwischen den einzelnen Minern gleich verteilt. Die Zeit, die benötigt wird, um den nonce-Wert zu finden, hängt also von der Komplexität ab: Je höher die Komplexität, desto länger dauert die Suche nach dem gewünschten nonce-Wert. Der PoW-Algorithmus repräsentiert das Konzept der Komplexität, das in dieser Blockchain verwendet wird. Der PoW-Algorithmus bietet Stabilität für die Blockchain und verhindert die Erstellung anderer Ketten, die die Transaktionshistorie beeinflussen können. Damit ein Teilnehmer als Erster seine Blöcke validieren kann, muss er jederzeit den Wert von nonce bestimmen und zwar schneller als jeder andere Netzwerkteilnehmer.

2.2.5 Vergütung

Einer der wichtigsten Punkte im Ethereum-System ist der Zahlungsprozess. Jede Rechenleistung innerhalb des Ethereum-Netzwerks ist gebührenpflichtig. Der Nominalwert dieser Zahlung wird als *Gas* bezeichnet.

Gas ist eine Messeinheit, die verwendet wird, um den Zahlungsbetrag für eine bestimmte Rechenleistung zu bestimmen. Der *Gas*-Preis ist die Menge an *ether*, die man für jede einzelne Rechenleistung ausgeben kann. Der Gaspreis wird in *gwei* gemessen (1 *ether* = 1,000,000,000 *gwei*).

Für jede Transaktion muss der Sender ein *Gas*-Limit und einen *Gas*-Preis festlegen. Der *Gas*-Preis und das *Gas*-Limit ist der Höchstbetrag in *gwei*, den der Sender bereit ist, für die Transaktion zu zahlen.

Beispielsweise legt ein Sender ein *Gas*-Limit von 50.000 *gwei* und einen *Gas*-Preis von 20 *gwei* fest. Dies bedeutet, dass der Absender bereit ist, nicht mehr als 50.000 * 20 *gwei* = 1.000.000.000.000.000 *wei* (0,001 *ether*) für die Transaktion auszugeben. Da *Miner* Transaktionen berechnen und verifizieren, sind sie diejenigen, die die *Gas*-Gebühr als Vergütung erhalten. Je höher der Transaktionspreis, desto attraktiver ist die Transaktion für *Miner* und desto schneller wird sie validiert. Die Bezahlung verhindert eine übermäßige Netzwerkbelastung.

2.2.6 Transaktionen

Das Ethereum ist ein Transaktionsstatus-System. Mit anderen Worten, Transaktionen, die zwischen verschiedenen Konten stattfinden, verändern den globalen Ethereum-Zustand [20].

Es gibt nur zwei Arten von Transaktionen: das Senden von Nachrichten und das Anlegen eines Smart Contract. Alle Transaktionen werden zu „Blöcken“ zusammengefasst. Die Ethereum-Plattform bietet die Möglichkeit, Protokolle zu führen, deren Zweck es ist, Informationen über verschiedene Transaktionen und Nachrichten aufzuzeichnen. Außerdem gibt es für Smart Contract auch die Möglichkeit, mit Hilfe eines „Ereignisses“ einen Eintrag in einem solchen Protokoll zu erstellen. Die Ausführung von Transaktionen im Ethereum ist ein ziemlich komplizierter Prozess. Eine sehr vereinfachte Beschreibung ist nachfolgend aufgeführt.

Nach einer bestimmten Validierung wird eine Gas-Anzahlung vom Senderkonto abgezogen und der nonce-Wert des Senderkontos um 1 erhöht. Danach beginnt die Ausführung der Transaktion. Während der aktuellen Transaktion werden die Zwischenstände im Ethereum verfolgt. Ein Zwischenstand ist notwendig, um die Informationen aufzuzeichnen, die während der aktuellen Transaktion gesammelt wurden. Der nächste Schritt besteht darin, die verschiedenen Berechnungen durchzuführen, die für die Durchführung der Transaktion selbst erforderlich sind.

Nachdem alle für die Durchführung der Transaktion erforderlichen Schritte abgeschlossen sind, wird die Transaktion beendet. Unverbrauchtes Gas wird an den Sender zurückgege-

ben, die Informationen werden im Protokoll gespeichert und die Zwischenstände werden gelöscht.

2.2.7 Smart Contract

Smart Contract sind Computerprotokolle, die Verträge abbilden oder überprüfen oder die Verhandlung oder Abwicklung eines Vertrags technisch unterstützen. Im einfachsten Fall sieht der Smart Contract aus wie ein normaler Vertrag mit streng definierten Bedingungen, der von verschiedenen Parteien unterzeichnet wird. Durch den Einsatz von Smart Contracts werden Vermittler ausgeschlossen; der Vertrag funktioniert nach bekannten Prinzipien – vorhersehbar, gleichberechtigt, transparent und unveränderbar.

Ein Smart Contract ist also ein Programm, das auf Ethereum Virtual Machine läuft. Der Smart Contract wird in einer Blockchain gespeichert, so dass alle Netzwerkmitglieder jeweils eine Kopie davon haben und der Smart Contract auch von allen Netzwerkmitgliedern identisch ausgeführt wird. Die Ausführung des Smart Contract erfordert die Zahlung einer bestimmten Provision.

Der Smart Contract im Ethereum hat einige wichtige Einschränkungen. Eine wichtige Einschränkung ist die fehlende Unterstützung für verschiedene Timer-Implementierungen und Verzögerungen bei der Codeausführung. Tatsächlich bedeutet dies, dass ein Smart Contract nicht „im Hintergrund“ funktionieren kann, er kann nur durch eine Transaktion des Benutzers oder eines anderen Smart Contract gestartet werden. Der zweite Aspekt ist, dass Smart Contracts nur Transaktionsdaten verarbeiten dürfen und nicht außerhalb der virtuellen Maschine verfügbar sind. Beispielsweise können Sie kein HTTP-Request zum Hochladen von Informationen senden.

Die Entwicklung von Smart Contracts im Ethereum erfolgt in der Programmiersprache Solidity. Für die Kompilierung wird das IDE Remix verwendet.

2.2.8 Geth

Die Interaktion mit Ethereum ist über eine Vielzahl von Clients möglich, von denen einige terminalbasiert, andere GUI-basiert oder hybride Lösungen sind. Geth ist ein Standardwerkzeug, das vom Ethereum-Team entwickelt wurde. Der Client ist in der Programmiersprache Go geschrieben und wird in der Regel aus dem Repository oder Quellcode

installiert. Geth selbst hat keine GUI und funktioniert nur vom Terminal aus. Mit geth kann man die Ethereum-Blockchain vollständig verwalten. Unter anderem ist Geth in der Lage:

- Verbindungen mit einer vorhandenen Blockchain zu erstellen
- eine eigene Blockchain zu erzeugen
- Mining durchzuführen
- Token von einem Konto auf ein anderes zu überweisen
- Smart Contract zu erstellen und auszuführen

Es ist sehr schwierig, alle Funktionen dieses Tools zu beschreiben. Es bleibt anzumerken, dass Geth auch ein Opensource-Projekt ist und sich in der aktiven Entwicklung befindet. Das Geth-Tool wird in den folgenden Kapiteln verwendet, um das private Blockchain bereitzustellen.

2.2.9 Swarm

Swarm stellt eine verteilte Speicherplattform dar. Swarm ist ein nativer Basisschichtdienst des ethereum web3-Stacks. Das Hauptziel von Swarm ist es, einen ausreichend dezentralen und redundanten Speicher für die öffentlichen Ethereum-Einträge bereitzustellen, insbesondere um Dapp-Code und Daten sowie Blockchain-Daten zu speichern und zu verteilen [9].

Swarm wurde entwickelt, um eine tiefe Integration mit der devp2p Multiprotokoll Netzwerkebene von Ethereum sowie mit der Ethereum-Blockchain für die Auflösung von Domainnamen, Service-Zahlungen und Content-Verfügbarkeit zu ermöglichen [17].

Swarm ist ein Peer-to-Peer-Netzwerk von Knoten, die verteilte digitale Dienste bereitstellen, indem sie Ressourcen (Speicher, Nachrichtenweiterleitung, Zahlungsabwicklungen) einander zur Verfügung stellen. Swarm bietet eine lokale HTTP-Proxy-API, die Dapps oder Kommandozeilen-Tools zur Interaktion mit Swarm verwenden können. Einige Module wie Messaging sind nur über die RPC-JSON API verfügbar. Swarm ist eine Sammlung von Knoten des devp2p-Netzwerks, von denen jeder die BZZ-URL-Schemata auf derselben Netzwerk-ID ausführt. Standardoperationen zum Hinzufügen oder Empfangen von Daten werden mit Hilfe einer einfachen http-Request durchgeführt [9]. Swarm erlaubt

es, einfach Daten in den Swarm hochzuladen und dann offline zu gehen. Solange andere Knoten verfügbar sind, sind auch die Daten verfügbar. Dies ist durch die Synchronisation möglich, bei der Knoten kontinuierlich verfügbare Daten untereinander weitergeben. Swarm ist eine persistente Datenstruktur, weshalb es keine Löschfunktion gibt [9].

2.3 IoT und Blockchain

Es gibt noch eine Vielzahl von Forschungsherausforderungen und damit offenen Fragen um IoT und Blockchain nahtlos miteinander zu verbinden, die untersucht werden müssen. Dabei lohnt es sich vielleicht zunächst, damit zu beginnen, welche Vorteile diese Integration bringen kann. Im Folgenden gilt es, einige Punkte genau vorzustellen, die Verbesserungen beinhalten, welche diese Integration mit sich bringen kann:

- Dezentralisierung und Skalierbarkeit: Der Wechsel von einer zentralen Architektur zu einer verteilten Peer-to-Peer beseitigt zentrale Fehler- und Engpasspunkte [49]. Dies trägt dazu bei, Szenarien zu verhindern, in denen einige mächtige Unternehmen die Verarbeitung und Speicherung der Informationen einer großen Anzahl von Geräten / Personen kontrollieren. Weitere Vorteile, die mit der Dezentralisierung der Architektur einhergehen, sind eine Verbesserung der Fehlertoleranz und der Skalierbarkeit des Systems.

- Identität: Mit einem gemeinsamen Blockchain-System können die Teilnehmer jedes einzelne Gerät identifizieren. Die Daten, die bereitgestellt und in das System eingespeist werden, sind unveränderlich und identifizieren eindeutig die tatsächlichen Daten, die von einem Gerät bereitgestellt sind. Darüber hinaus kann die Blockchain eine vertrauenswürdige verteilte Authentifizierung und Autorisierung von Geräten für IoT-Anwendungen bieten [13].

- Autonomie: Die Blockchain-Technologie ermöglicht Anwendungsmerkmale der nächsten Generation und die Entwicklung intelligenter autonomer Vermögenswerte und "Hardware as a Service" [12]. Mit der Blockchain sind Geräte in der Lage, ohne die Beteiligung von Servern miteinander zu interagieren. IoT-Anwendungen könnten von dieser Funktionalität profitieren, um geräteunabhängige und entkoppelte Anwendungen bereitzustellen.

- Zuverlässigkeit: IoT-Informationen bleiben unveränderlich und können über die Zeit über eine Blockchain verteilt werden [26]. Die Teilnehmer des Systems sind in der

Lage, die Authentizität der Daten zu überprüfen und können sicherstellen, dass sie nicht manipuliert werden.

- Sicherheit: Blockchain kann den Austausch von Gerätenachrichten als Transaktionen behandeln, die durch Smart-Contracts validiert werden, um so die Kommunikation zwischen den Geräten zu sichern [33].

- Sicheres *DevOps*: Durch die Nutzung des sicheren, unveränderlichen Speichers der Blockchain kann sichergestellt werden, dass keine unbemerkten Änderungen am Quellcode vorgenommen werden und dieser den Zielgeräten sicher zur Verfügung gestellt werden kann [45]. Hersteller können Zustände und Updates mit höchster Zuverlässigkeit verfolgen [33]. IoT-Middleware könnte diese Funktionalität nutzen, um IoT-Geräte sicher zu aktualisieren.

Die obigen Punkte zeigen, dass das Internet der Dinge im Verbund mit Blockchain, viele Probleme lösen kann und sich sich neue Einsatzmöglichkeiten der Technologien in diesem Kontext entwickeln.

2.3.1 Blockchain-IoT Interaktionen

Ein weiterer Aspekt betrifft die Kommunikation zwischen der zugrundeliegenden IoT-Infrastruktur. Bei der Integration der Blockchain muss entschieden werden, wo der Prozess dieser Interaktionen stattfindet: innerhalb des IoT, in einem hybriden Design mit IoT und Blockchain oder in einem Prozess der durch die Blockchain verarbeitet wird. Fog Computing [24] hat auch das IoT mit der Aufnahme einer neuen Schicht zwischen Cloud Computing und IoT-Geräten revolutioniert und könnte diese Integration erleichtern.

- IoT-IoT (Abb. 2.5a): Dieser Ansatz könnte der schnellste in Bezug auf Latenz und Sicherheit sein, da er offline arbeiten kann. IoT-Geräte müssen in der Lage sein, miteinander zu kommunizieren, was in der Regel mit Erkennungs- und Routingmechanismen verbunden ist. Nur ein Teil der IoT-Daten wird in der Blockchain gespeichert, während die IoT-Interaktionen ohne Verwendung der Blockchain stattfinden. Dieser Ansatz wäre nützlich in Szenarien mit zuverlässigen IoT-Daten, in denen die IoT-Interaktionen mit geringer Latenzzeit stattfinden.

- IoT-Blockchain (Abb. 2.5b): Bei dieser Vorgehensweise laufen alle Interaktionen durch Blockchain und ermöglichen eine unveränderliche Aufzeichnung der Interaktionen. Sie stellt sicher, dass alle gewählten Interaktionen nachvollziehbar sind, da

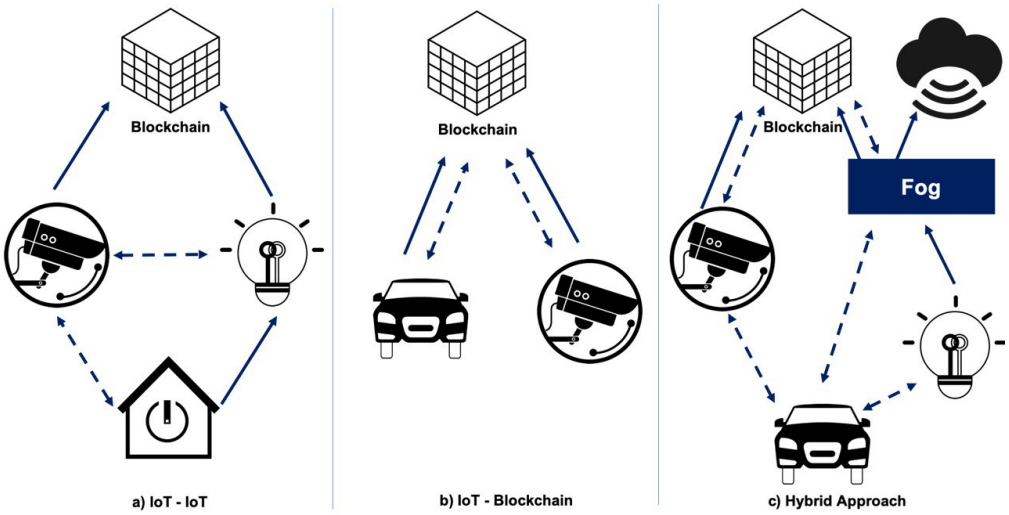

Abbildung 2.5: Blockchain-IoT Interaktion

ihre Details in der Blockchain abgefragt werden können und zudem die Autonomie der IoT-Geräte erhöht wird. Dennoch würde die Aufzeichnung aller Interaktionen in der Blockchain eine Erhöhung der Bandbreite und der Daten mit sich bringen, was eine der bekannten Herausforderungen in der Blockchain ist. Andererseits sollten alle mit diesen Transaktionen verbundenen IoT-Daten auch in der Blockchain gespeichert werden.

- Hybridansatz (Abb. 2.5c): Der Hybridansatz ist ein Design, bei dem nur ein Teil der Interaktionen und Daten in der Blockchain stattfinden, der andere interagiert direkt zwischen den IoT-Geräten. Eine der Herausforderungen bei diesem Ansatz ist die Frage, welche Interaktionen durch die Blockchain vollzogen werden sollen. Eine perfekte Orchestrierung dieses Ansatzes wäre der beste Weg, beide Technologien zu integrieren, da er die Vorteile der Blockchain und die Vorteile der Echtzeit-Interaktionen nutzen würde. Bei diesem Ansatz könnte Fog Computing ins Spiel kommen und sogar Cloud Computing, um die Grenzen der Blockchain und des IoT zu ergänzen [37, 10].

In einem typischen IoT-Deployment werden ressourcenbeschränkte Geräte als Endknoten verwendet, die mit einem Gateway kommunizieren. Dieses Gateway ist für die Weiterleitung von Sensordaten an obere Schichten (Cloud oder Server) zuständig. Bei der Integration von Blockchain, wenn Endknoten mit der Blockchain interagieren müssen, kann in IoT-Geräten eine kryptographische Funktionalität bereitgestellt werden. Dies ist der

Schlüssel zur Realisierung der IoT-Autonomie, die mit einer anspruchsvolleren Hardware und höheren Rechenkosten verbunden ist. Die Integration von Gateways in diese Lösungen ist plausibel, ähnlich wie bei traditionellen Deployments, dennoch sind die Vorteile der Verwendung von Blockchain auf diese Weise geringer.

Um die Lücke zwischen IoT und Blockchain zu schließen, bilden große IT-Unternehmen Allianzen, wie z.B. Trusted IoT Alliance [5]. Außerdem gibt es auf dem Markt immer mehr Geräte mit integrierten Blockchain-Funktionen[43]. Daher könnte diese Art von Gerät in Smart Homes installiert werden und Teil eines Fog-Computer-Ökosystems sein. Auf einigen dieser IoT-Geräte kann das Mining durchgeführt werden, bleibt aber ineffizient. Mining ist zu einer spezialisierten Aufgabe für bestimmte Hardware (ASIC) geworden und es wäre nicht ratsam, es auf gängigen IoT-Geräten zu implementieren. Das ist der Hauptgrund, warum es wenig Bemühungen für das Thema Mining für IoT-Geräte gibt. Im Allgemeinen besteht noch Forschungsbedarf, um eine breite Integration von IoT-Geräten als Blockchain-Komponenten zu ermöglichen.

Vollständige Knoten müssen die gesamte Blockchain (derzeit 210 GB in Bitcoin [29] und 132 GB in Ethereum [47]) für eine vollständige Validierung von Transaktionen und Blöcken speichern, so dass ihr Einsatz durch die umfangreiche Kapazitätsnutzung bei IoT-Geräten sehr eingeschränkt ist. Wie bereits erwähnt, wäre der Mining auf dem IoT-Gerät aufgrund seiner spezifischen Anforderungen nutzlos. Das Konsensusprotokoll könnte vereinfacht werden, um die Einbindung von IoT-Geräten zu erleichtern, da dies der Sicherheit der Blockchain-Implementierung im Wege steht. Dies könnte in “consortium blockchain” eingesetzt werden, in denen die Konsensusprotokolle reduziert werden. Beim Einsatz von Light-Knoten wird die Authentizität der Transaktionen überprüft, ohne die gesamte Blockchain herunterladen zu müssen und können weiterhin im IoT als Teil des Blockchain-Netzwerks verwendet werden. Diese muss immer mit vollständigen Knoten gesichert werden, um Transaktionen und Blöcke zu validieren. Dennoch bieten viele Blockchain noch keine Unterstützung für Light-Knoten. In jedem Fall könnte die Blockchain als externer Dienst genutzt werden, um einen sicheren und zuverlässigen Speicher bereitzustellen.

Eine klare Alternative zur Integration der Blockchain mit dem IoT ist die Integration zwischen dem IoT und Cloud Computing [8]. Diese Integration wurde in den letzten Jahren genutzt, um die Einschränkungen des IoT bei Verarbeitung, Speicherung und Zugriff zu beheben. Cloud Computing bietet jedoch in der Regel eine zentralisierte Architektur, die im Gegensatz zur Blockchain die zuverlässige gemeinsame Nutzung mit vielen Teil-

nehmer aufwendig macht. Fog Computing zielt darauf ab, das Computing zu verteilen und den Endgeräten näher zu bringen, indem es einem verteilten Ansatz wie der Blockchain folgt. Dabei können leistungsfähigere Geräte wie Gateways integriert werden, die dann als Blockchain-Komponenten wiederverwendet werden können. Daher könnte Fog Computing die Integration des IoT mit der Blockchain vereinfachen.

2.3.2 Herausforderungen bei der Blockchain-IoT Integration

Der folgende Abschnitt untersucht die wichtigsten Herausforderungen, die bei der Anwendung der Blockchain-Technologie auf den IoT-Bereich zu bewältigen sind. Die Integration der Blockchain-Technologie mit dem IoT ist nicht trivial. Blockchain wurde ursprünglich für ein Internet-Szenario mit leistungsfähigen Computern entwickelt, allerdings ist dies trotz immer weit entfernt von der IoT-Realität. Einige der identifizierten Herausforderungen werden in diesem Abschnitt vorgestellt.

Speicherkapazität und Skalierbarkeit

Wie bereits erwähnt, sind die Herausförderungen an Speicherkapazität und Skalierbarkeit der Blockchain immanent, und im Kontext von IoT-Anwendungen erscheinen die Kapazitäts- und Skalierbarkeitseinschränkungen dieser Herausforderungen noch größer. Doch auch wenn die Blockchain für IoT-Anwendungen ungeeignet erscheinen mag, gibt es bei näherer Betrachtung, Möglichkeiten, diese Einschränkungen zu verringern oder ganz zu vermeiden [50].

Im IoT, wo Geräte Gigabyte an Daten in Echtzeit erzeugen können, stellt diese Einschränkung ein großes Hindernis für die Integration der Blockchain dar. Es ist bekannt, dass einige aktuelle Blockchain-Implementierungen nur wenige Transaktionen pro Sekunde verarbeiten können, somit ist auch angesprochen, dass dies ein möglicher Engpass für das IoT sein könnte. Darüber hinaus ist die Blockchain nicht dafür ausgelegt, große Datenmengen, wie die im IoT erzeugten, zu speichern. Eine Integration dieser Technologien sollte sich denn diesen Herausforderungen stellen.

Derzeit sind viele IoT-Daten gespeichert, doch nur ein begrenzter Teil ist verwertbar. Für die Menge der nicht verwertbaren Daten wurden Techniken zum Filtern, Normalisieren und Komprimieren von IoT-Daten mit dem Ziel, diese zu reduzieren, vorgeschlagen. Die

Datenkompression kann die Übertragung, die Verarbeitung und die Speicherung von erzeugten IoT-Daten vereinfachen. Blockchain (und insbesondere sein Konsensusprotokoll) könnten auch angepasst werden, um die Bandbreite zu erhöhen und die Latenzzeiten ihrer Transaktionen zu verringern [11].

Sicherheit

IoT-Anwendungen müssen sich mit Sicherheitsproblemen auf verschiedenen Ebenen auseinandersetzen. Die Herausforderung besteht darin, dass sich wachsende Komplexität, geringe Performanz und hohe Heterogenität der Geräte gegenüberstehen. Darüber hinaus umfasst das IoT-Szenario eine Reihe von sicherheitsrelevanten Eigenschaften/Lücken wie Mobilität, drahtlose Kommunikation etc.

Die zunehmende Anzahl von Angriffen auf IoT-Netzwerke und deren schwerwiegende Auswirkungen machen es umso notwendiger, ein IoT mit anspruchsvollen Sicherheitsfeatures zu schaffen. Viele Experten sehen in der Blockchain die vielversprechendste Technologie, um die dringend benötigten Sicherheitsverbesserungen im IoT zu erreichen. Eine weitere der größten Herausforderungen bei der Integration des IoT mit Blockchain ist jedoch die Zuverlässigkeit der IoT-Daten. Blockchain kann zwar sicherstellen, dass die Daten in der Chain unveränderlich sind und ihre Transformationen identifiziert werden können. Wenn die Daten bereits beschädigt übermittelt werden, bleiben diese in der Blockchain fehlerhaft.

Die Stabilität der IoT-Architektur wird durch viele Faktoren wie Umgebung, Teilnehmer, Vandalismus und den Ausfall der Geräte beeinflusst. Manchmal funktionieren die Geräte selbst richtig, doch ihre Sensoren und Aktoren nicht. Zusätzlich zu diesen Situationen gibt es viele Bedrohungen, die das IoT beeinflussen können, wie *eavesdropping*, *Denial-of-Service* und viele weitere [42].

Darüber hinaus ist es oft aufwendig, Geräte einzelnd zu aktualisieren, besonders bei globalen IoT-Systemen. Daher sollten im IoT Laufzeit-Upgrades und Rekonfigurationsmechanismen platziert werden.

Die IoT- und Blockchainintegration kann sich auch auf die IoT-Kommunikation negativ auswirken [21]. Derzeit nutzen IoT-Anwendungsprotokolle wie CoAP und MQTT andere Sicherheitsprotokolle als TLS oder DTLS, um eine sichere Kommunikation zu gewährleisten. Diese Protokolle sind komplex und erfordern zusätzlich eine zentralisierte

Verwaltung, typischerweise mit PKI. Im Blockchain-Netzwerk würde jedes IoT-Gerät seine eigene GUID (Global Unique Identifier) und ein asymmetrisches Schlüsselpaar haben, die nach der Verbindung mit dem Netzwerk installiert werden.

Smart contracts

Smart Contracts wurden als "Killer-Feature"der Blockchain-Technologie identifiziert. Das IoT könnte von der Verwendung von Smart Contracts profitieren, wobei die Art und Weise, wie sie in IoT-Anwendungen eingesetzt werden, vielfältig ist [8].

Auf der einen Seite erfordert die Arbeit mit Smart Contracts den Einsatz von speziellen Einheiten, die reale Daten zuverlässig bereitstellen. Durch die Instabilität des IoT, ist es möglich, dass die Validierung dieser Smart Contracts gefährdet wird. Darüber hinaus könnte der Zugriff auf mehrere Datenquellen diese Contracts überlasten. Heutzutage sind Smart Contracts verteilt und dezentralisiert, wobei sie sich keine Ressourcen teilen und einen großen Rechenaufwand zu bewältigen. Mit anderen Worten, die Ausführung von Smart Contracts erfolgt in nur einem singulären Knoten, während die Codeausführung gleichzeitig von mehreren Knoten durchgeführt wird. Diese Verteilung wird nur für den Validierungsprozess durchgeführt, aber es findet keine Verteilung von Smart Contract Aufgaben statt.

Big Data hat zwar die gleichzeitige Verarbeitung großer Datenmengen ermöglicht, so dass Wissen aus großen Datensätzen extrahiert werden kann, was bisher sehr schwierig war, allerdings nutzen die Smart Contract bei der Integration von IoT mit Blockchain ihren verteilten Charakter noch nicht, um die Verarbeitungsmöglichkeiten zu vergrößern, die in anderen Paradigmen (Big Data und Cloud Computing) schon bereitgestellt sind und auch im IoT benötigt werden. Desweiteren sollten Smart Contracts auch die Heterogenität und die Einschränkungen des IoT berücksichtigen. Dies könnte ein Discoverymechanismus, der die sofortige Einbindung von Geräten zur Verfügung stellt, ermöglichen, was diese Anwendungen leistungsfähiger machen würde.

Die Integration der beiden Technologien birgt noch viele weitere Herausforderungen. Einige von ihnen beziehen sich nur auf bestimmte Plattformen, andere solcher sind von allgemeinerer Technik. Aus Blockchain-Problemen entstehen automatisch Probleme für jede Blockchain-basierte Anwendung. Die Probleme von Blockchain mit ihrer Skalierbarkeit und die gleichzeitigen Anforderungen bei Latenz und Durchsatz von IoT [15, 31] waren und sind weiterhin von großem Forschungsinteresse. Diese Bachelorarbeit will sich

auf die Probleme der Skalierbarkeit der Problemgröße [48] von Blockchain-basierten IoT-Backend konzentrieren.

3 Experiment

In diesem Kapitel werden der Aufbau und die Struktur des Experiments beschrieben. Unterkapitel 3.1 beschäftigt sich mit dem Design des hier verwendeten Blockchain-basierten Backends und den Werkzeugen, die für seine Untersuchung benötigt werden. Gegenstand des Unterkapitels 3.2 ist eine schrittweise Beschreibung der Konfiguration und Bereitstellung des Backends und seiner Umgebung sowie anderer Elemente des Experiments. Es folgt Unterkapitel 3.3, in welchem der Ablauf des Experiments erläutert wird.

3.1 Testnetzwerk

Dieses Unterkapitel beschreibt das Design des Blockchain-basierten Backends, die verwendete Hardware und das Design des Testtool, um den Aufbau des Experiments darzustellen.

3.1.1 Backend Design

Das Design des betrachteten Backends ist in einer Studie von Özyılmaz und Yurdakul (2018) mit dem Namen „Designing a blockchain-based IoT infrastructure with Ethereum, Swarm and LoRa“ dargestellt [30]. Sie schlagen eine Architektur vor, in der auf allen Geräten Ethereum und Swarm Knoten laufen, die dort in einem gemeinsamen Netzwerk miteinander verbunden sind. Alle Datenpakete, die in das System gelangen, werden an einen lokalen Proxy-Server gesendet. Es ist dieser lokale Server, genannt Smart Proxy, der die Daten vom Paketförderer empfängt und als Vermittler fungiert, um Daten in die Blockchain-basierte Infrastruktur zu übertragen. Schließlich vervollständigen Swarm und Ethereum Clients die Softwarearchitektur (Abb. 3.1). Im Rahmen des Experiments werden die Daten innerhalb Smart Proxy generiert.

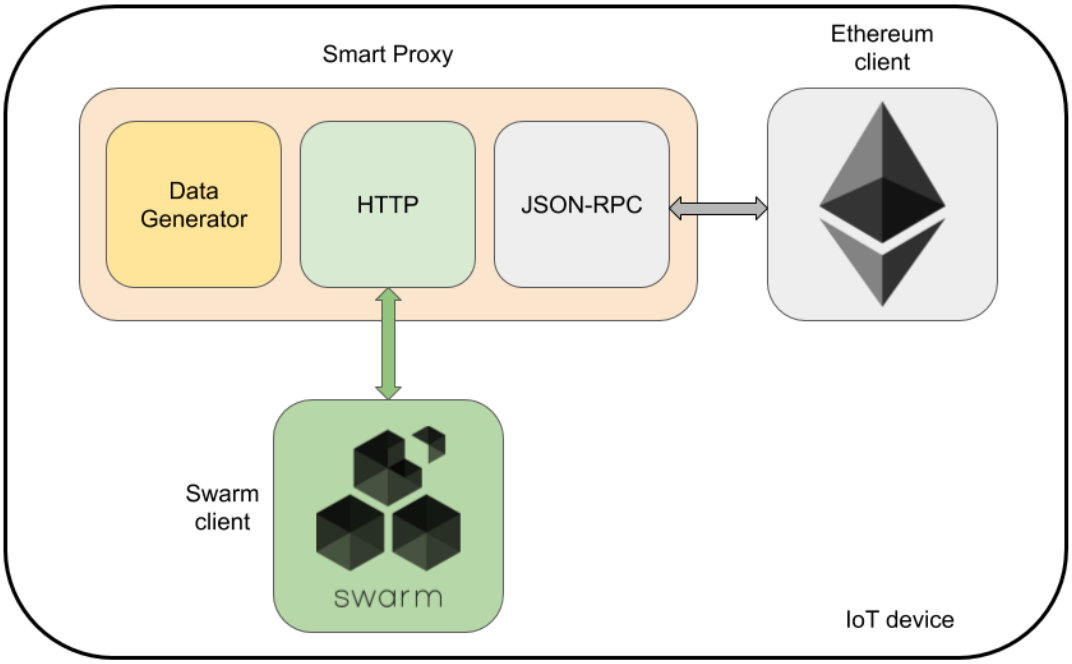

Abbildung 3.1: IoT-Backend Architektur

Nachdem das Gerät Daten erzeugt (oder von einem Dritten erhalten) hat, werden diese über HTTP in das Swarm-Dateispeichernetzwerk geschoben. An diesem Punkt wird ein File-Hash empfangen.Derselbe File-Hash wird in Zukunft für den Zugriff auf diese bestimmten Daten verwendet werden.

Ein Smart Proxy kann über seine JSON-RPC-Schnittstelle mit dem Ethereum-Client kommunizieren. Um jedoch eine echte Interaktion mit dem Ethereum-Netzwerk zu ermöglichen, muss bereits ein Smart Contract installiert sein. Nachdem dieser in Bytecode kompiliert wurde, wird ein Smart Contract wie jede andere Transaktion versendet, die von Minern bearbeitet wird [siehe Kapitel Ref]. Ist das Mining ders Smart Contracts abgeschlossen, wird seine Adresse und seine binäre Anwendungsschnittstelle (ABI) zur Interaktion mit dem Ethereum-Client verwendet. Der Code des betrachteten Smart Contract ist in Anhang A.3 aufgelistet.

Der in diesem Projekt entwickelte und betrachtete Smart Contract enthält ein Event und sechs Funktionen, die in den Listing 3.1 dargestellt sind.

```
1 function is_device_present (address device_id) public constant
    returns (bool result);
```

```
2 function get_device_count() public constant returns (uint count
    );
3 function get_device_at_index (uint index) public constant
    returns (address device_address);
4 function get_device_timestamps (address device_id) public
    constant returns (uint[] ts);
5 function get_device_data (address device_id, uint ts) public
    constant returns (string hash);
6 function set_device_data (address device_id, string filehash)
    public returns (uint index, uint ts);
7 event log_action (address indexed device_id, uint index, uint
    ts, string filehash);
```

Listing 3.1: Smart Contract Interface

Alle Funktionen mit Ausnahme der set_device_data()-Funktion sind konstantundverändern den Smart Contract-Zustand nicht. Nur die set_device_data()-Funktion fügt Daten zur Blockchain hinzu, daher sollte für diese eine Blockchain-Transaktion eingerichtet werden.

Die Auflistung A.1 zeigt das aktuelle Codefragment, das angibt, wie Geräteidentifikationen, Zeitstempelwerte und Swarm-Hashes miteinander verbunden sind. IoT-Geräte und ihre gespeicherten Hashes in der Blockchain können einfach gesichtet und mit den Funktionen get_device_timestamp() und get_device_data() aufgerufen werden.

```
1 // IoT device data
2 struct device_data {
3     // link for detecting devices
4     uint index;
5     // blockchain timestamps stored swarm hashes
6     uint[] timestamps;
7     // map timestamp values to swarm file hashes
8     mapping(uint => string) filehashes;
9 }
10 // map device id's to their data (one data per id)
11 mapping(address => device_data) private device_logs;
12 // keep a separate device id array of all received id's
13 address[] private device_index;
```

```
14 // event to log action
15 event log_action (address indexed device_id, uint index, uint
       ts, string filehash);
```

Listing 3.2: Datenstruktur

Die Funktion set_device_ data() ist die eigentliche Smart Contract Funktion, die einen Swarm-Hash auf den aktuellen Blockzeitstempel abbildet. Sobald neue Daten hinzugefügt wurden, wird das Log Action Event geschaltet und alle Peers, die dieses Event erreicht, erhalten einen *Callback*. Somit lassen sich zwei Hauptszenarien unterscheiden - Datenspeicherung und Datenerfassung (Abb. 3.2). Diese beiden Grundfunktionen sind die Mindestanforderungen für die Implementierung des IOT-Backends. Ist dies gegeben, können sie verwendet werden, um ein typisches Szenario einer Benutzertransaktion zum sequentiellen Schreiben und Lesen von Daten anzuzeigen.

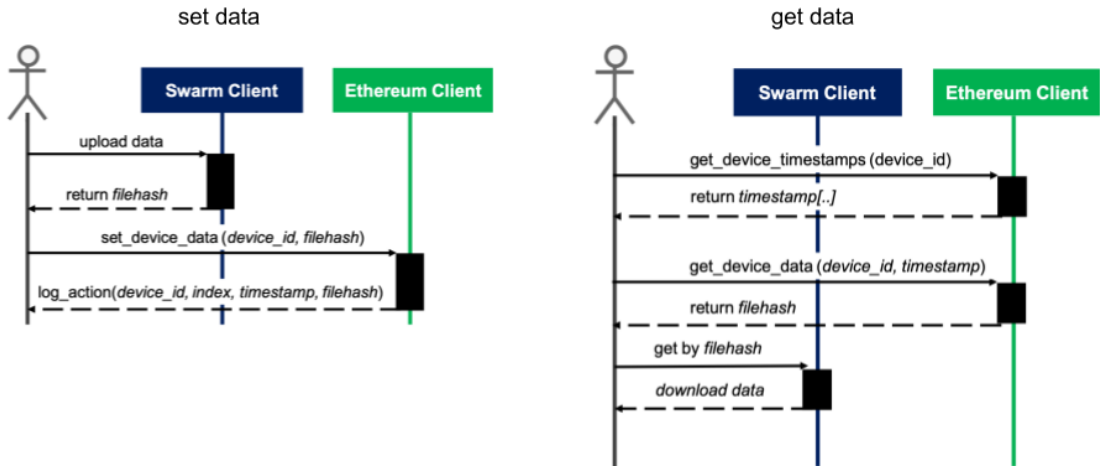

Abbildung 3.2: Datenspeicherung und Datenerfassung

3.1.2 Testnetzwerk Design

Das Testnetzwerk besteht aus zwei Endgeräten und einem Server. Als Endgeräte wurden Raspberry Pi 2b und Raspberry Pi 3b ausgewählt, da diese einfach zu konfigurieren sind und sie - dank der Ausbildung der ARM-Architektur - alle notwendigen Tools und Technologien für das jeweilige Projekt unterstützen. Es sind zwei sehr ähnliche, doch auch unterschiedliche Geräte, die im Folgenden in der Tabelle 3.1 nebeneinander gestellt werden.

Tabelle 3.1: Beschreibung der Geräte

Gerät	Prozessor (CPU)	Arbeitsspeicher (RAM)
Raspberry Pi 2B	Broadcom BCM2836 4 x 900 MHz	1 GB SDRAM
Raspberry Pi 3B	Broadcom BCM2837 4 x 1,2 GHz	1 GB SDRAM
Google VM	Intel Xeon 8 x 2.20GHz	8 GB RAM

Der Raspberry Pi 3B hat einen 1,2 GHz Quadcore-Prozessor der nun auch 64-Bit unterstützt; im Vergleich zum Raspberry Pi 2A ein Sprung um 4 x 300 MHz. Die RAM-Belegung in den Geräten ist gleich. Die Rolle beider Geräte besteht darin, Daten zu erstellen und an das Backend zu senden. Auf den Geräten wird aufgrund ihrer Leistung kein Mining ausgeführt, dennoch werden alle Daten (Ethereum, Swarm) darauf gespeichert. Somit betreiben beide Raspberry's den Full-Ethereum-Knoten ohne Mining und den normalen Swarm-Knoten.

Um das Mining in einem Testnetzwerk durchzuführen, wurde ein Server mit einem 8-Kern-Prozessor und 8 GB RAM ausgewählt (s. Tabelle 3.1). Desweiteren wird im Rahmen des Experiments konventionelles CPU-Mining ohne die Beteiligung von GPUs eingesetzt. Somit ist der Server mit einem Full-Miner-Knoten und einem Swarm-Boot-Knoten ausgestattet. Das gesamte Netzwerkdesign ist in Abbildung 3.3 dargestellt.

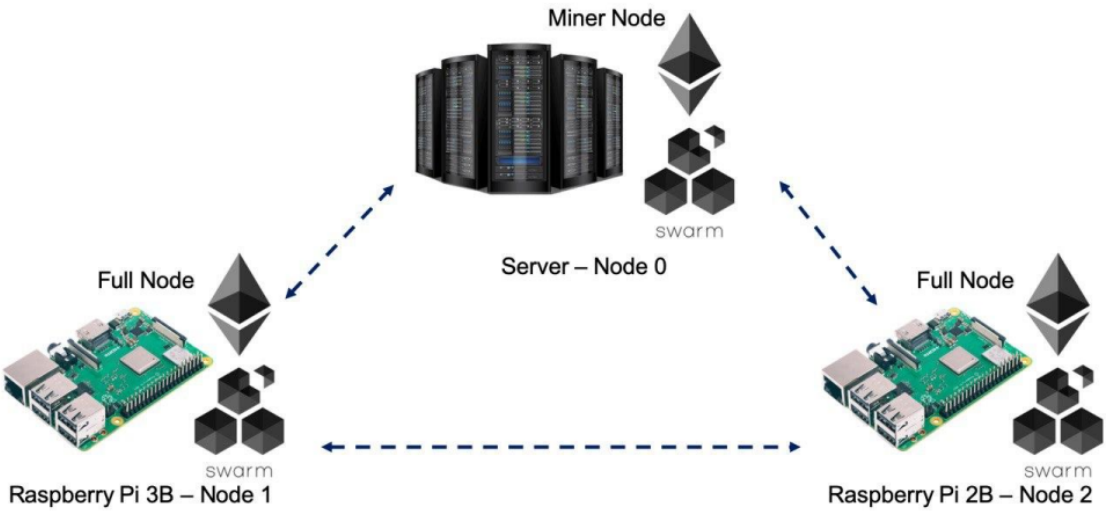

Abbildung 3.3: Testnetzwerkplan

Eine detaillierte Beschreibung der Testnetzwerkkonfiguration wird in Abschnitt 3.2 erläutert.

3.1.3 Testtool

Für die Untersuchung der Skalierbarkeit des jeweiligen IoT-Backends, wurde ein Testtool entwickelt, das die sequentiellen Schreib- und Lesetransaktionen prüft. Das Testprogramm ist in folgender Weise angelegt: Es automatisiert Smart Proxy, d.h. es erstellt die API für Kommunikation mit Ethereum / Smart Contract und Swarm. Es erzeugt Datenlast auf dem IoT-Backend. Es misst und protokolliert den Transaktionsvorgang.

Da der Kernpunkt des Projekts die Kommunikation des Testprogramms mit der Ethereum-Blockchain ist, wurde bei der Auswahl der Programmiersprache die Verwendungsmöglichkeiten der Bibliotheken zur Kommunikation mit der Ethereum-Blockchain geprüft.

Der De-facto-Standard in diesem Bereich ist die Web3.js. Web3.js ist eine Sammlung von Bibliotheken, die ermöglicht, mit einem lokalen oder entfernten Ethereum-Knoten über eine HTTP-, WebSocket- oder IPC-Verbindung zu kommunizieren. Web3 ist in vielen Programmiersprachen implementiert, darunter: Python, Haskell, Java, Scala, PHP.

Die ursprüngliche Version von Web3.js ist mit javascript implementiert. Dieser Tatsache wegen, sowie der Popularität von Javascript bei der Ethereum-DApp-Entwicklung, fiel die Wahl auf node.js als Haupttechnologie bei der Entwicklung des Testtools.

Das Testtool ist dahingehend konfigurierbar, dass es auf unterschiedlichen Geräten anwendbar ist. Die Konfigurationsdateien sind nachfolgend aufgelistet (Anhang A.2). Die Architektur des Testprogramms ist in Diagramm 3.4 dargestellt: Die Test-Aufgaben (TransaktionSet) wurden auf der Application-Ebene erstellt und bestimmen die Anzahl der Transaktionsaufrufe pro Zeiteinheit. Eine wichtige Rolle auf dieser Ebene spielt der Scheduler. Der Scheduler ist aus dem Cron-Scheduler-Format entworfen und startet die Controller-Schicht in bestimmten Intervallen.

Auf der Controller-Ebene übernimmt der TransactionController die Test-Aufgabe und führt eine sequentielle Schreib- und Lesetransaktion durch. Außerdem protokolliert der TransactionController alle Transaktionsvorgänge mit Hilfe von Logitem. Hierbei erfolgt ein sequentielles Aufrufen der Funktionen der Service-Schicht, in der sich EthereumService und SwarmService befinden. Folgende Schritte werden auf dem EthereumService durchgeführt, bevor die Transaktion ausgeführt werden kann:

1. Erstellen einer Verbindung zur Ethereum-Knoten
2. Ethereum-Konto entsperren

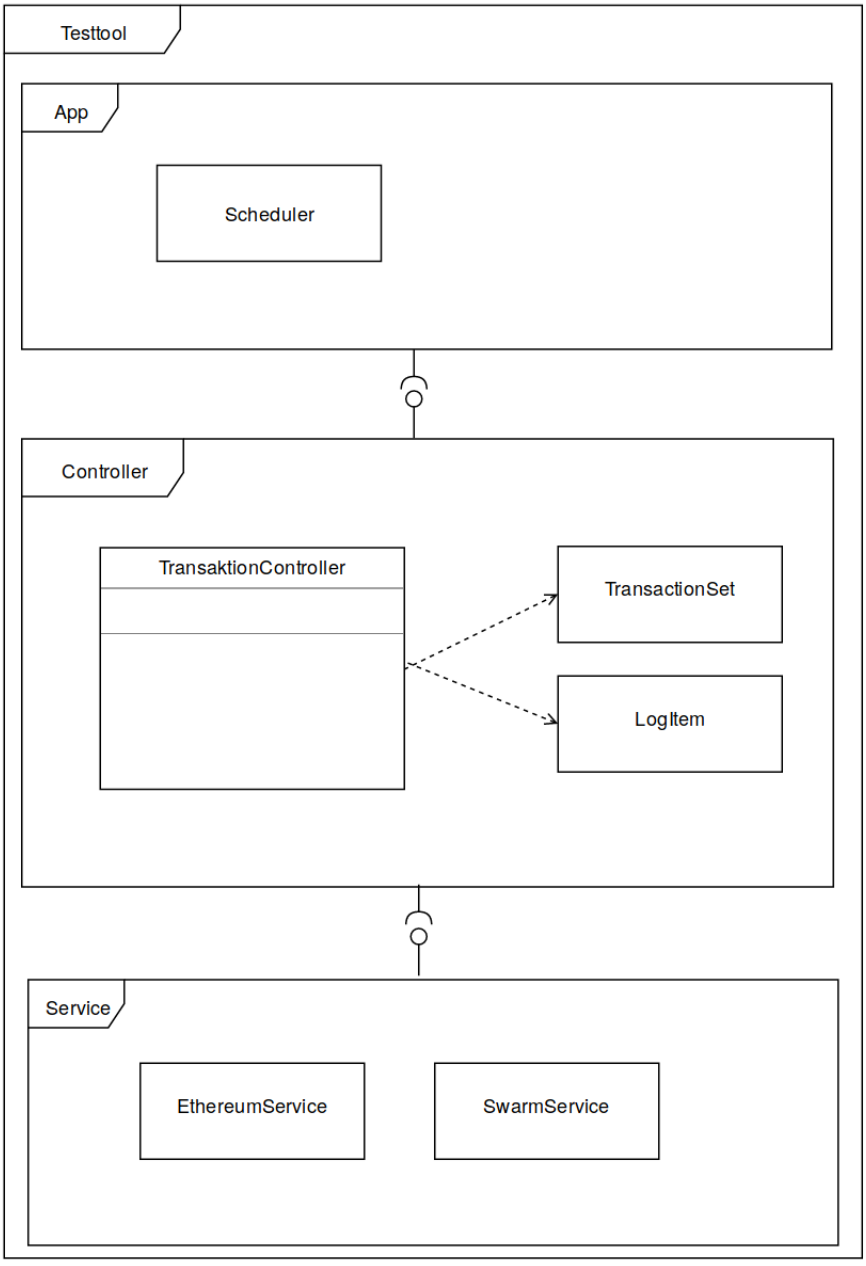

Abbildung 3.4: Testtool: Komponentendiagramm

3. Smart Contract Initialisierung

Der Inhalt der Transaktion selbst kann wie folgt beschrieben werden:

1. Die Daten in Swarm schreiben (SwarmService)
2. Swarm-Hash in Ethereum mit Smart Contract schreiben (EthereumService)
3. Letzter Daten-Zeitstempel aus der Ethereum Smart Contract lesen (EthereumService)
4. Swarm-Hash aus der Ethereum Smart Contract mit Hilfe der Device ID und Daten-Zeitstempel lesen (EthereumService)
5. Die Daten aus Swarm mit Hilfe des Swarm-Hash lesen (SwarmService)

Der SwarmService kommuniziert mit seinem Knoten über http-Requests - POST zum Schreiben und GET zum Lesen - mit Hilfe der axios-Bibliothek. Der EthereumService nutzt die bereits erwähnte web3-Bibliothek und ruft die im Abschnitt XYZ beschriebenen Smart Contract-Methoden mit Hilfe von RPC auf.

Das Protokoll misst die Verzögerung aller fünf oben genannten Teile der Transaktion. Darüber hinaus wird die Äquivalenz der zurückgegebenen Daten mit den entsendeten, sowie die Größe der Ethereum-Blockchain und Swarm-Größe protokolliert. Die Ergebnisse werden in einer neuen Datei im Projektordner gespeichert.

3.2 Testnetzwerk Setup

Im folgenden Kapitel wird die Einrichtung der Hardware, der Blockchain, der Testumgebung und aller anderen für das Experiment erforderlichen Elemente beschrieben.

3.2.1 Vorbedingungen

Der erste Schritt im Setup-Prozess für alle Netzwerkteilnehmer ist die Auswahl eines Betriebssystems. Das Betriebssystem Ubuntu 18.04.2 LTS wird auf dem Server installiert. Es beinhaltet alle Standardwerkzeuge wie ssh-Client, die im Rahmen des Experiments notwendig sind. Beide Raspberry Pi sind mit dem Betriebssystem Raspbian Stretch Lite

ausgestattet. Diese minimierte Version ohne Frontend ermöglicht es einem, die Ressourcen auf das Backend selbst zu konzentrieren.

Einer der wichtigsten Schritte ist die Installation von Clients für Ethereum und Swarm. In dieser Arbeit wird der offizielle Ethereum-Client geth verwendet, der bereits in Abschnitt 2.2.8 erwähnt wurde. Zuerst müssen einige Tools und insbesondere vorkompilierte Go-Binärdateien auf allen Geräten heruntergeladen und installiert werden.

```
$ sudo apt install -y curl
$ sudo apt install -y build-essential
$ sudo apt install -y screen
```

Die Go-Versionen für den Server und Raspberry Pi unterscheiden sich in der Architektur. Für den Server:

```
$ curl -O https://storage.googleapis.com/golang
  /go1.9.linux-amd64.tar.gz
```

Für beide Raspberry Pi:

```
$ curl -O https://storage.googleapis.com/golang
  /go1.9.linux-armv6l.tar.gz
```

Als nächstes müssen Go-Binärdateien installiert werden:

```
$ sudo tar -C /usr/local -xzf go$VERSION.$OS-$ARCH.tar.gz
```

Am Ende wird die PATH-Variable zu /etc/profile oder $HOME/.profile hinzugefügt.

```
$ export PATH=$PATH:/usr/local/go/bin
$ sudo reboot
```

Anschließend kann man die Clients installieren. Zu diesem Zweck wird das Go-Ethereum-Projekt aus dem offiziellen Projekt-Repository kopiert und mit dem Befehl *make* installiert.

```
$ git clone https://github.com/ethereum/go-ethereum.git
$ cd go-ethereum
$ make geth
$ make swarm
$ sudo cp build/bin/geth /usr/local/bin/geth
$ sudo cp build/bin/swarm /usr/local/bin/swarm
```

3.2.2 Ethereum

Zuerst müssen Konten auf allen Geräten erstellt werden und die Pfade zu den einzelnen Ethereum-Verzeichnissen angeben werden, in denen die Blockchain gespeichert wird.

```
$ geth account new --datadir /home/ethnode
```

Sobald das Passwort festgelegt ist, zeigt die Konsole die Konto-ID an.

Jede Blockchain beginnt mit dem Genesis-Block und alles weitere wird auf diesem aufgebaut, wobei sich jeder folgende Block auf den vorherigen Block bezieht. Um eine private Blockchain zu erzeugen, muss man einen eigenen Genesis-Block erstellen. Der Genesis-Block im Ethereum ist nichts anderes als eine Konfigurations-Json-Datei. Der Genesis-Block (genesis.json) für diese Bachelorarbeit ist im Anhang A.1 aufgelistet.

```
$ geth --datadir "~/ethnode" init genesis.json
```

Mit dem folgenden Befehl wird der erste Miner-Knoten auf dem Server gestartet.

```
$ geth --rpc --rpcport "8001" --rpccorsdomain "*"
  --datadir "~/ethnode" --port "30303" --rpcapi
  "db,eth,net,web3" --identity "zero" --networkid 733
  --mine --minerthreads 1
```

Als Netzwerk-ID (–networkid) kann eine beliebige Zahl angegeben werden. Dadurch können andere Knoten mit dem privaten Netzwerk verbunden werden. Zusätzlich ist auf diesem Knoten der HTTP-RPC-Server mit mehreren aktiven Schnittstellen aktiviert. Es ist wichtig zu beachten, dass nur dieser Knoten für das Mining verwendet wird (–mine).

Jetzt ist es möglich, normale Knoten auf den IoT-Geräten zu initialisieren. Die Knoten auf Raspberry Pi starten wie folgt:

```
$ geth /usr/local/bin/geth --rpc --rpcport "8002"
  --rpcaddr "0.0.0.0" --rpccorsdomain "*" --datadir
  "~/ethnode" --port "30303" --rpcapi
  "db,eth,net,web3" --identity "one" --networkid 733
```

Es müssen verschiedene RPC-Ports, aber die gleiche Netzwerk-ID verwendet werden. Nun ist das private Ethereum-Netzwerk betriebsbereit.

3.2.3 Swarm

Die Storage-Plattform Swarm erfordert keine komplizierte Konfiguration. Der erste Schritt besteht darin, einen Boot-Knoten zu erstellen. Sie wird auf dem Server erstellt. BZZKEY ist eine gültige Konto-ID, die bereits bei der Einrichtung des Ethereum-Kontos generiert wurde.

```
$ swarm --bzzaccount $BZZKEY --datadir "~/ethnode"
  --ens-api "/home/ethnode/geth.ipc" --bzznetworkid 666 &
```

Nun kann man den zweiten und dritten Knoten auf den beiden Raspberry Pi starten. Diese verwenden den ersten Knoten als Boot-Knoten.

```
$ swarm --bzzaccount $BZZKEY --datadir "~/bether" --ens-api
  "/home/ethnode/geth.ipc" --bzznetworkid 666
  --bootnodes "enode://$ENODE_ID@$IP_ADDR:30399"
```

Die Funktionsfähigkeit des Netzwerks kann sofort überprüft werden. Mit dem nächsten Befehl werden die Daten in Swarm hochgeladen. Nach der Befehl-Ausführung erscheint der Hash-Wert für die entsprechende Datei in der Konsole.

```
$ curl -H "Content-Type: text/plain" --data-binary
  "test-data" http://localhost:8500/bzz:/
```

Bereits hochgeladene Daten können nun von einem beliebigen Knoten mithilfe des eben erzeugten Hash abgerufen werden.

```
$ curl http://localhost:8500/bzz:/hash-of-test-data
```

Wenn Swarm korrekt funktioniert, kann mit dem nächsten Schritt fortgefahren werden.

3.2.4 Smart Contract

Der für die Implementierung des Blockchain-basierten Backends erforderliche Smart Contract ist im Anhang A.3 enthalten. Für die Bearbeitung und Kompilierung von Smart Contract wird der IDE Remix verwendet. Nach der Kompilierung des Smart Contract werden zwei Dateien erstellt: vendor.js und event.js. Das sind die Smart Contract-Dateien, die in das Ethereum-Netzwerk aufgenommen werden müssen. Dazu verbindet man sich zuerst mit der geth-Konsole und loggt sich ein.

```
$ geth --datadir "/home/ethnode" attach
  ipc:/home/ethnode/geth.ipc console
$ > personal.unlockAccount(eth.coinbase)
```

Jetzt lässt sich Smart Contract und Event Listener herunterladen.

```
$ > loadScript('vendor.js');
$ > loadScript('event.js');
```

In der Konsole erscheint eine Meldung, dass Smart Contract erfolgreich hinzugefügt wurde, sowie die Adresse in der Ethereum Blockchain. Diese Adresse wird später für den Betrieb des Testtool benötigt. Um sicherzustellen, dass Smart Contract erfolgreich hinzugefügt wurde, ist folgender Befehl auszuführen, damit die Anzahl der angeschlossenen Geräte sichtbar wird.

```
$ > vendor.get_device_count();
```

Smart Contract wird einmalig auf dem Server hinzugefügt und steht danach allen Netzwerkmitgliedern zur Verfügung. Wenn Smart Contract erfolgreich hinzugefügt wurde, kann man eine Transaktions durchführen, welche Daten im Backend speichert und abfragt.

3.2.5 Testtool

Das Testtool ist, wie oben erwähnt, in der Programmiersprache *Javascript* implementiert. Es erfordert die Installation von *Node.js*, einer Softwareplattform auf Basis der V8-Engine, die (*JavaScript* in Maschinencode übersetzt). Die Installation auf dem Server erfolgt mit einem Befehl:

```
$ sudo apt install nodejs
```

Bei der Installation auf Raspberrys muss die richtige ARM-Version (6,7,8) von *Node.js* ausgewählt werden.

```
$ wget https://nodejs.org/dist/v10.16.0/
      node-v10.16.0-linux-armv7l.tar.xz
$ tar xvf node-v10.16.0-linux-armv7l.tar.xz
$ cd node-v10.16.0-linux-armv7l/
$ sudo cp -R * /usr/local/
```

Nach erfolgreicher Installation kann man das Testprogramm heruntergeladen werden:

```
$ git clone https://github.com/artyapple/ba-eth-sa.git
```

Im Anschluss muss die Konfigurationsdatei geändert.

```
$ cd ./ba-eth-sa
$ nano ./config/default.json
```

Nach diesen den Installationsschritten müssten alle Informationen zur Verfügung stehen, die für die Konfiguration notwendig sind. Das Programm wird wie folgt gestartet:

```
$ npm install
$ npm start
```

Zu diesem Zeitpunkt ist die Ausführung eines Testprogramms ist (noch) nicht sinnvoll. An dieser Stelle ist es notwendig, die Aufgaben für das Testtool zu erstellen, im nächsten Kapitel wird dies eingehender dargestellt.

3.3 Experimentablauf

Wie bereits in Kapitel 2.3 erwähnt, analysiert diese Arbeit die Skalierbarkeit der Problemgröße des Blockchain-basierten IoT-Backends. Somit möchte sie eine Aussage darüber treffen, wie sich ein IoT-Backend bei der Ausführung einer Transaktion (bestehend aus sequentiellem Schreiben und Lesen) in Abhängigkeit vom Wachstum des Problems, verhält.

Die Anzahl der Transaktionen ist ein einfacher und verständlicher Wert, der das System beeinflusst. Bezogen auf diese Problemgröße, wird ermittelt, auf welche Art bei steigender Anzahl der Transaktionen, die Transaktionskosten ebenfalls steigen. Der Aspekt der Latenz soll hierbei von primären Interesse sein.

Zu diesem Zweck wird im Rahmen des Experiments eine lineare Datenlast erzeugt. Die Daten, die das Testtool für die Transaktion generiert, haben folgende Struktur:

```
1  '{' + new Date().getTime() + '}'
```

Tabelle 3.2: Experiment: Datenlast pro Stunde

Stunde	Transaktionen pro St. (pro Gerät)	Transaktionen pro St. (insgesamt)	kumulative Summe
1	5	15	15
2	5	15	30
3	5	15	45
4	10	30	75
5	10	30	105
6	10	30	135
7	15	45	180
8	15	45	225
9	15	45	270
10	20	60	330
11	20	60	390
12	20	60	450
13	25	75	525
14	25	75	600
15	25	75	675
16	30	90	765
17	30	90	855
18	30	90	945
19	35	105	1050
20	35	105	1155
21	35	105	1260
22	40	120	1380
23	40	120	1500
24	40	120	1620
25	45	135	1755
26	45	135	1890
27	45	135	2025
28	50	150	2175
29	50	150	2325
30	50	150	2475
31	55	165	2640
32	55	165	2805
33	55	165	2970
34	60	180	3150
35	60	180	3330
36	60	180	3510

Somit überträgt jede Transaktion ein String von 15 Zeichen oder 15 Bytes Daten.

Die Last, die von den einzelnen Geräten ausgeht, beginnt bei einem Wert von 5 Transaktionen pro Stunde und steigt jeweils in Schritten von 5 weiter an. Die endgültige, vom Einzelgerät ausgehende Anzahl, beläuft sich schließlich auf 60 Transaktionen. Für viele IoT-Szenarien (wie z.B. das Energie- oder Umweltmonitoring) macht ein Testrahmen mit einer Höchstbelastung von mehr als einer Transaktion pro Minute, wenig Sinn.

Darüber hinaus wird jeder Problemwert dreimal wiederholt. Dies soll dazu beitragen, die Daten zu normalisieren und Verzerrungen durch andere Störgrößen (wie z.B. Netzwerkprobleme) zu minimieren. Der Gesamtplan des Experiments ist nach Stunden aufgeschlüsselt, in Tabelle 3.2 aufgeführt.

Damit die Transaktionen gleichmäßiger über die ganze Stunde verteilt werden, werden die Geräte zu unterschiedlichen Zeitpunkten gestartet. Node0 startet zur vollen Stunde, der Node1 mit einer Verzögerung von 20 Minuten zur vollen Stunde und Node2 mit einer weiteren Verzögerung um 40 Minuten (zur vollen Stunde). Dies wird durch Testtool-Scheduler-Einstellungen erreicht.

Nach der Ausführung der Transaktion sollten die mit den Eingangsdaten identischen Daten empfangen worden sein. Auf diese Weise kann die Error-Rate des betrachteten Backends bestimmt, die Ursachen der Fehler analysiert werden und der Zusammenhang mit der wechselnden Datenlast festgestellt werden. Die letzte Aufgabe des Überwachungsplans stellt die Gesamtgröße der Ethereum Blockchain und des Swarm dar.

Trotz der geringen Komplexität des Minings (difficulty=1), haben die ersten Vorversuche bereits eine Miningdauer von bis zu 300 Sekunden ermittelt. Wie zuvor erwähnt, wirft der PoW-Algorithmus, ein Thema außerhalb dieser Arbeit, im Allgemeinen viele Fragen im Zusammenhang mit dem Internet der Dinge auf.

Das Ethereum erhöht ständig den Komplexitätsgrad des Mining [52]. Für das Experiment ist eine möglichst kurze Miningdauer kritisch. Der Mining-Wert mit der niedrigsten Komplexität ist, die „1“. Um die Komplexität statisch auf 1 zu setzen, muss der consensus.go Quellcode geändert werden [25]. Bei etwa Linie 311 von consensus.go gibt es eine Funktion namens CalcDifficulty. Die Auflistung 3.3 zeigt, wie diese Funktion die Komplexität des Minings durch den Aufruf einer von 4 anderen Funktionen einstellt.

Listing 3.3: Ethereum: consensus.go - Original

```
func CalcDifficulty(config *params.ChainConfig, time uint64, parent *types.Header) *big.Int
{
        next := new(big.Int).Add(parent.Number, big1)
        switch {
        case config.IsConstantinople(next):
                return calcDifficultyConstantinople(time, parent)
        case config.IsByzantium(next):
                return calcDifficultyByzantium(time, parent)
        case config.IsHomestead(next):
                return calcDifficultyHomestead(time, parent)
        default:
                return calcDifficultyFrontier(time, parent)
        }
}
```

Für das Experiment wurde der gesamte Code innerhalb des Rumpfes der CalcDifficulty-Funktion entfernt und nur eine einzige Return-Anweisung hinzugefügt, damit Ethereum kontinuierlich auf seinem niedrigstmöglichen Komplexitätsgrad verweilt. Die modifizierte CalcDifficulty-Funktion ist in Listing 3.4 dargestellt.

Listing 3.4: Ethereum: consensus.go - verändert

```
func CalcDifficulty(config *params.ChainConfig, time uint64, parent *types.Header) *big.Int
{
        return big.NewInt(1)
}
```

4 Ergebnisse

Dieses Kapitel beschreibt die Ergebnisse des in Kapitel 3.3 beschriebenen Experiments. Der Abschnitt 4.1 konzentriert sich auf das Thema Latenz. Im Folgenden wird die Error-Rate des betrachteten Backends bestimmt und die Ursachen der Fehler werden analysiert. Zum Abschluss werden in Kapitel 4.3 die Größe der Ethereum-Blockchain und des Swarm analysiert.

Die Dauer des Experiments betrug, wie geplant, 36 Stunden, wobei während der 30. Stunde Netzwerkprobleme auftraten. Die Geräte wurden für eine Weile vom Netzwerk getrennt, was zum Datenverlust führte. Dadurch ging jeweils der dritte Versuch bei Problemgröße 50 für node0 und node2 verloren. Trotz des Verlustes von ca. 2,5 % der Daten waren zwei weitere Versuche bei einer Problemgröße von 50 erfolgreich, wodurch der Verlust vernachlässigbar wurde.

4.1 Latenz

Das Hauptinteresse hinter dem Experiment ist die Latenz. Latenz bezeichnet die Verzögerung zwischen der Eingabe in ein System und der erwünschten Ausgabe.

Tabelle 4.1: Latenz Übersicht

ID	Gerät	Latenz (5 Schritte) ms	Latenz (Insgesamt) ms
node0	Google VM	25031	26072
node1	Raspberry Pi 3B	25271	26425
node2	Raspberry Pi 2B	25242	26391

Wie bereits in 3.1 erwähnt, besteht die Transaktion aus 5 Schritten, die im Wesentlichen das Aufrufen des Smart Contract oder die Kommunikation mit Swarm beinhalten. Im Rahmen des Experiments ist es notwendig, den 6. Schritt zu berücksichtigen. Hierbei

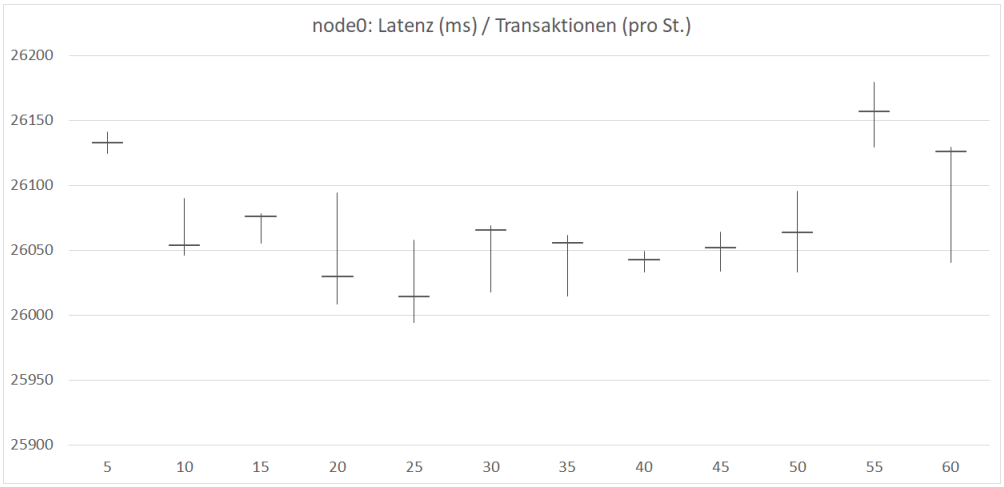

Abbildung 4.1: node0: Latenz

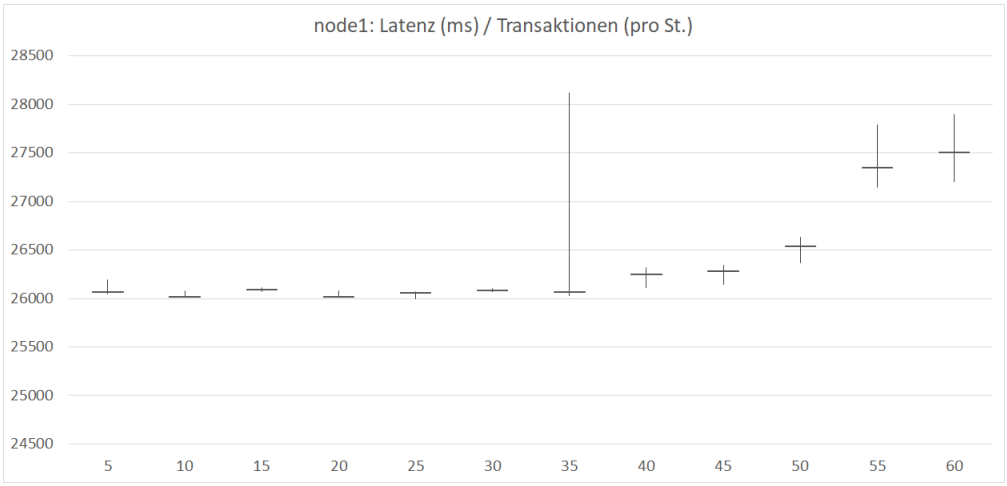

Abbildung 4.2: node1: Latenz

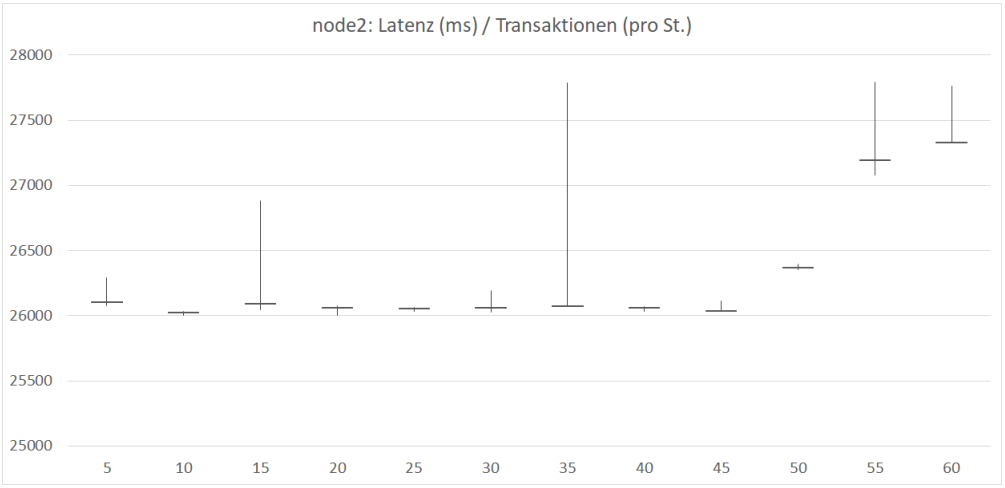

Abbildung 4.3: node2: Latenz

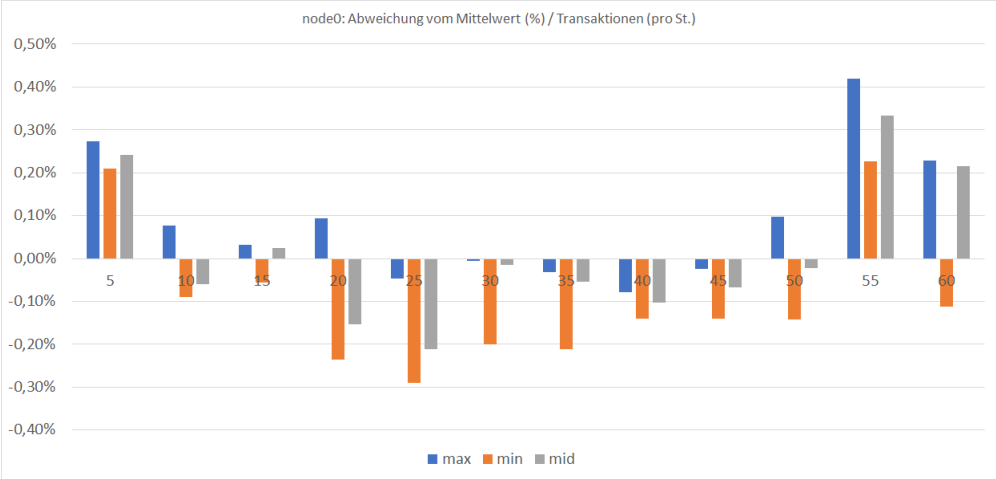

Abbildung 4.4: node0 - Latenz: Abweichung von Mittelwert

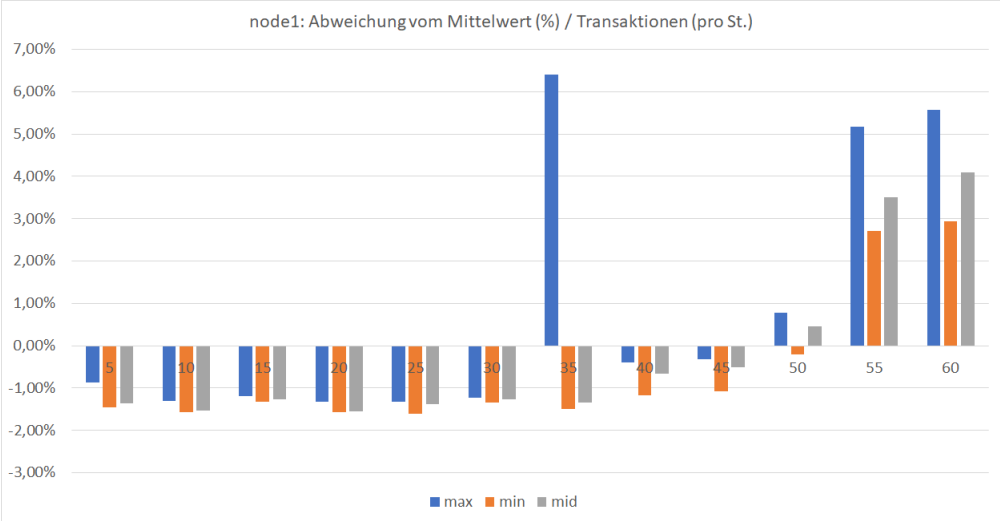

Abbildung 4.5: node1 - Latenz: Abweichung von Mittelwert

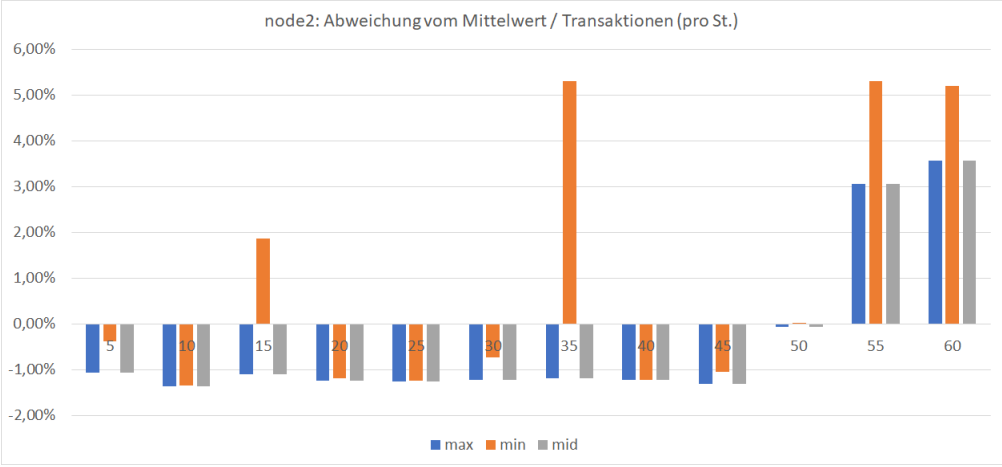

Abbildung 4.6: node2 - Latenz: Abweichung von Mittelwert

geht es um die Protokollierung, Erzeugung und Validierung von Testdaten und anderen damit verbundenen Systemkosten. Die durchschnittliche Latenz pro Gerät ist in Tabelle 4.1 dargestellt. node1 und node2 zeigen identische Latenzsraten. node0 (Server) ist im Durchschnitt etwa 300 ms schneller als andere Geräte.

Die Abbildungen 4.1, 4.2 und 4.3 zeigen die Latenz in Abhängigkeit von der Problemgröße für node0, node1 und node2. Die Diagramme verwenden ein Intervall, in dem die Ergebnisse von drei Versuchen für jeden Wert liegen.

Die node0-Latenzwerte konzentrieren sich zwischen 26000 ms und 26200 ms und weisen eine Abweichung von etwa 50 ms für jede Problemgröße auf. node1 zeigt eine gleichmäßige Latenz für Problemgröße 5 bis 30: ca. 26050 ms. Bei Problemgröße 35 liegt die Spitze bei über 28000 ms. Ab Problemgröße 40 steigt die durchschnittliche Latenz von 26000 ms auf 27500 ms. node2 ist node1 sehr ähnlich mit dem einzigen Unterschied, dass es einen zusätzlichen Spitzenwert bei Größe 15 gibt.

Die Abbildungen 4.4, 4.5 und 4.6 zeigen die Latenzabweichung von den Mittelwerten für jedes Gerät. node0 zeigt eine sehr gleichmäßige Verteilung der Werte; die Abweichung vom Durchschnitt beträgt weniger als 1%. Im Gegensatz dazu weisen node1 und node2 ein breites Spektrum von Werten auf, etwa -1,5% bis 6% des Durchschnitts.

Auf Abbildung 4.7 sieht man alle Geräte inklusive ihrer Durchschnittswerte im Verhältnis zur Problemgröße. node1 und node2 zeigen eine deutliche Latenzsteigerung gegenüber node0 ab Problemgröße 55. Die Entwicklung der Latenz-Durchschnittswerte über alle Geräte (Abb. 4.8) hinweg sprechen für eine langsame Latenzsteigerung bei steigender Anzahl von Transaktionen pro Stunde.

Um diese Steigerung zu verstehen, sollte man berücksichtigen, was mit den einzelnen Teilen der Transaktion passiert ist. Der größte Teil der Transaktion ist die set_data Operation auf dem Ethereum Smart Contract, die im Wesentlichen das Mining beinhaltet. Wie aus der Grafik 4.9 ersichtlich wird, führt das Mining bei einer Steigerung der Transaktionen pro Stunde zu keiner Latenzsteigerung. Im Gegenteil, geht es im Laufe der Zeit sogar mit einer gewissen Beschleunigung einher. Die Latenzwerte liegen zwischen 24200 ms und 25100 ms mit einer Ausnahme von zwei Spitzen in der 8. und 19. Stunde des Experiments. Bei näherer Betrachtung der Geräte fällt auf, dass die Werte von node1 und node2 im Durchschnitt um 200-300 ms geringer als bei node0 ausfallen, obwohl der gesamte Mining-Prozess nur bei node0 stattfindet. Dies ist erstaunlich, da node0 das

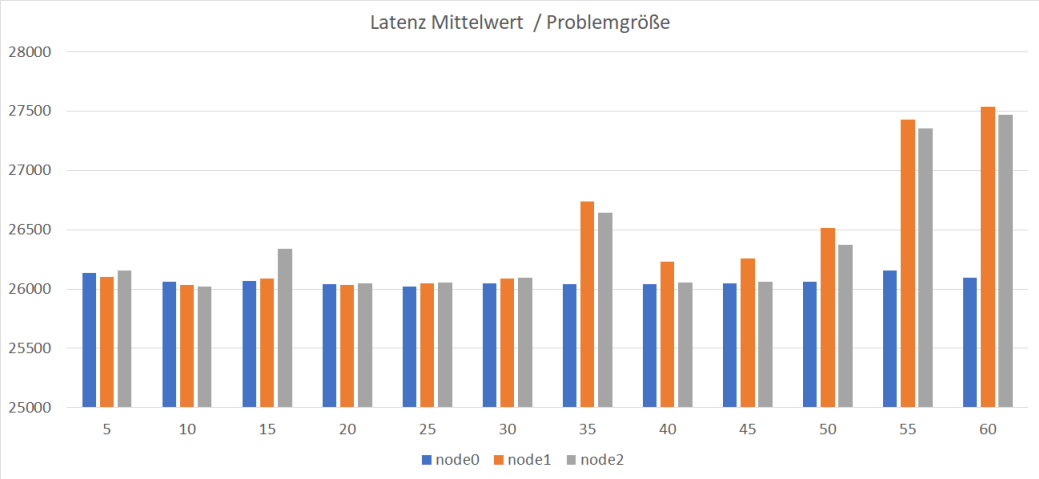

Abbildung 4.7: Latenz: Mittelwert pro Gerät

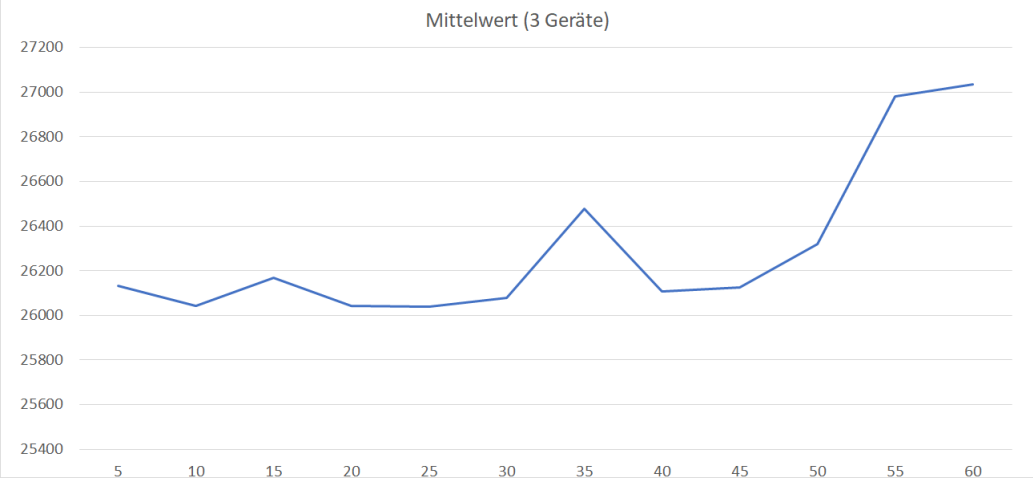

Abbildung 4.8: Latenz: Mittelwert

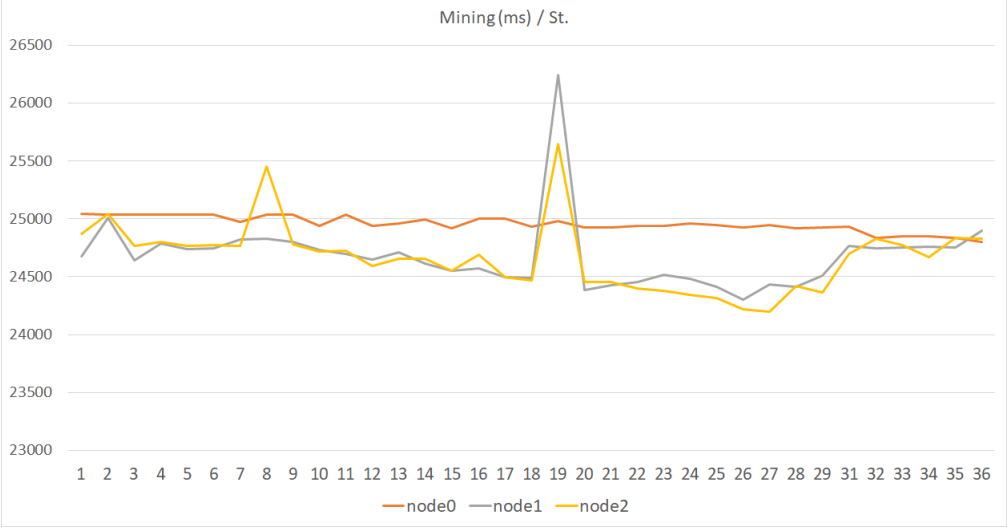

Abbildung 4.9: Latenz: Mining

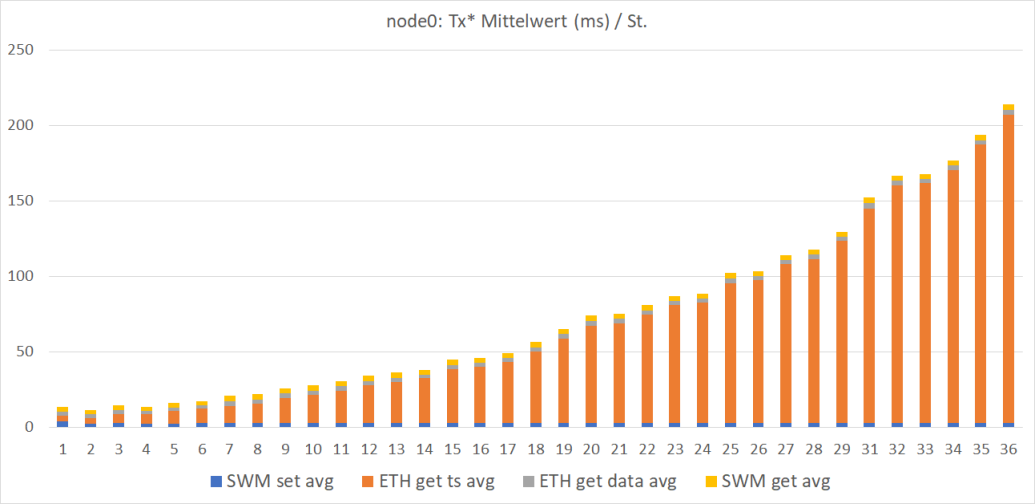

Abbildung 4.10: node0 - Latenz: Transaktionsbestandteile

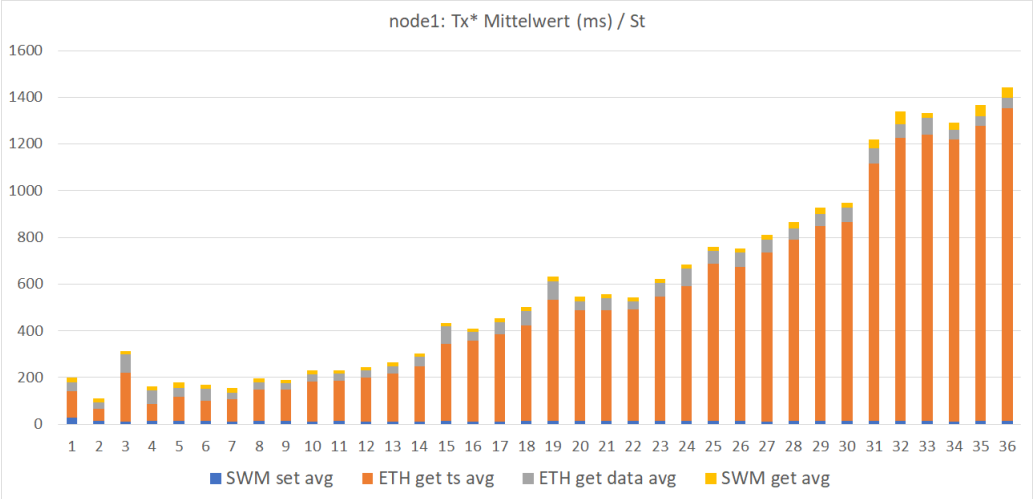

Abbildung 4.11: node1 - Latenz: Transaktionsbestandteile

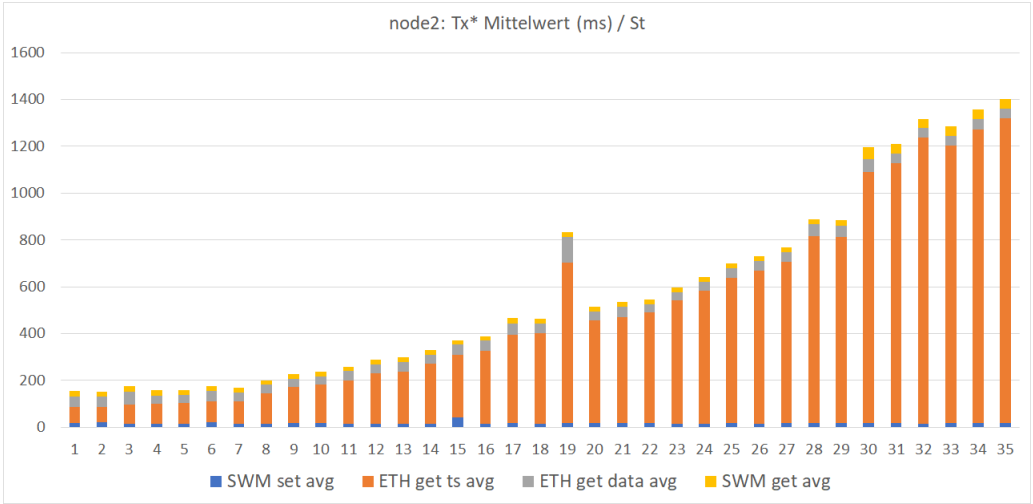

Abbildung 4.12: node2 - Latenz: Transaktionsbestandteile

Mining selber durchführt und keine Übergabe auf andere Mining-Knoten machen muss, wie node1 und node2.

Um den Rest der Transaktion einfacher zu betrachten, ist es sinnvoll, das bereits untersuchte Mining auszuschließen. Dann gibt es immer noch einen Getter und einen Setter für Swarm und auch get_data_timestamp, get_data Aufrufe auf Smart Contract.

Die Abbildungen 4.10, 4.11 und 4.12 stellen die Latenz dieser Transaktionsschritte für 3 Geräte dar. Wie aus diesen Diagrammen zu erkennen ist, hat die Kommunikation mit Swarm und get_data Aufruf von Smart Contract eine konstante Verzögerung im Bereich 2-5 ms für node0 und 15-50 ms für beide Raspberry's.

Der Aufruf der get_data_timestamp Operation auf einem Smart Contract zeigt ein schnelles, lineares Latenzwachstum im Laufe der Zeit. Bei node0 erhöht sich der Wert um mehr als das Fünfzigfache von 4 ms auf 204 ms. node1 steigt von 50 ms auf 1339 ms und node2 von 67 ms auf 1302 ms. Offensichtlich steigt die Latenz dieser Funktion im Verhältnis zur Menge der Daten, die im betrachteten Ethereumbasierten-Backend hinzugefügt werden (Abb. 4.13).

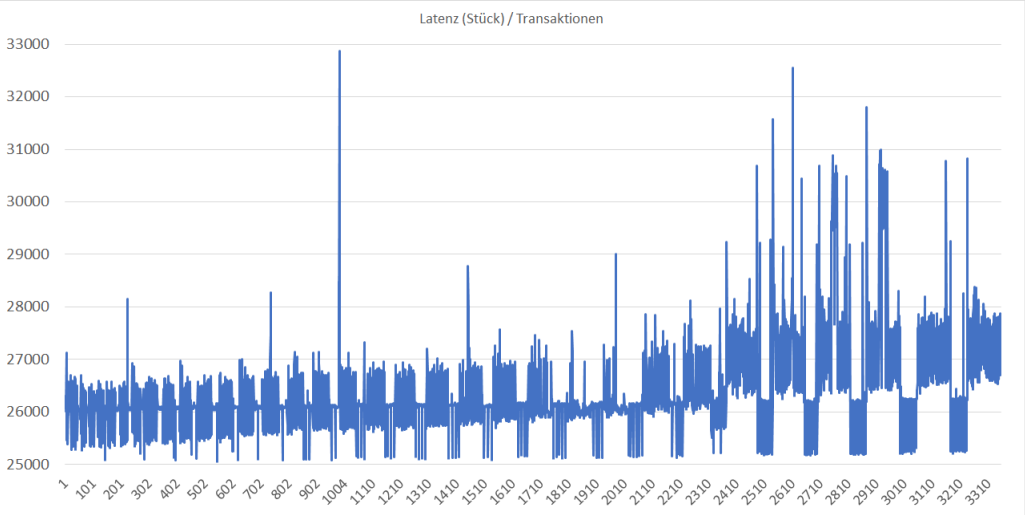

Abbildung 4.13: Latenz: Stückkosten

Das Wachstum ist auf die Art des verwendeten Smart Contract zurückzuführen. Die Zeitstempel werden in einem einfachen Array von Type unsigned integer gespeichert. Eine Suche nach einem Element in einem solchen Array hat eine Komplexität von $\mathcal{O}(n)$. Je mehr Daten dem System hinzugefügt werden, desto länger dauert die Suche nach dem notwendigen Zeitstempel. Das Hinzufügen eines Elements kann die verwendete Zeit zur

Suche um bis zu 1 ms erhöhen. So kann sich nach dem Hinzufügen weiterer 30.000 Daten die Gesamttransaktionslatenz verdoppeln.

4.2 Fehlerrate

Um die Fehlerrate zu untersuchen, wurden die Eingangsdaten mit den Ausgangsdaten verglichen. Äquivalente Daten wurden als korrekt berücksichtigt. Stimmen die Ausgangsdaten nicht mit den Eingangsdaten überein, liegt ein Fehler vor. Die Fehlerrate ist ein wichtiger Parameter, der deutlich macht, wie zuverlässig das Backend, abhängig von der Problemgröße, funktioniert. Hohe Fehlerwerte machen die Verwendung eines solchen Backends für den Anwender sinnlos.

Tabelle 4.2: Fehlerrate Übersicht

Gerät ID	Transaktionen	Fehler	Fehlerrate
node0	1120	9	0,80%
node1	1170	29	2,48%
node2	1120	28	2,50%
Summe	3410	66	1,94%

Wie aus Tabelle 4.2 zu erkennen ist, hat das Backend während des Experiments insgesamt 3410 Transaktionen von allen Geräten verarbeitet. Von diesen 3410 Transaktionen waren 66 Transaktionen fehlerhaft. Dies entspricht 1,94% der Gesamtzahl der Transaktionen. Die Fehlerrate von node0 ist niedriger als der eben genannte Durchschnitt, aber mit 0,80% immer noch hoch. node1 und node2 weisen eine mehr als drei Mal so hohe Fehlerrate wie node0 auf: 2,48% und 2,50%.

Die Abhängigkeit der Fehleranzahl von der Problemgröße ist in Diagramm 4.14 dargestellt. Wie aus diesem Diagramm ersichtlich wird, beträgt die Fehlerrate 0% für Problemgrößen bis zu 50. Bei 55 Transaktionen pro Stunde steigt der Fehlerwert auf 3,30% für den Server/Google VM und 7,88% für node1 und node2. Mit einer Problemgröße von 60 setzt sich die Steigerung für node1 und node2 fort: 8,89% für node1 und 8,33% für node2. node0 zeigt mit einer Fehlerrate von 2,22% bei einer Problemgröße von 60 ein etwas besseres Ergebnis als bei einer Problemgröße von 55.

Aus diesen Daten folgt, dass die Fehlerrate mit der Anzahl der Transaktionen pro Stunde steigt. Im Rahmen des Experiments wurden die an das Backend gesendeten Transaktio-

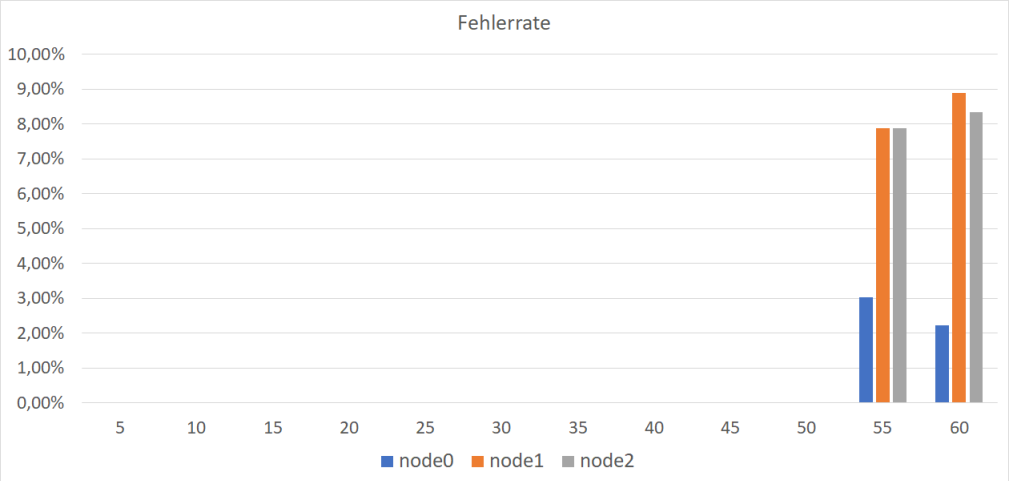

Abbildung 4.14: Fehlerrate

nen gleichmäßig über eine Stunde verteilt. Deshalb begannen die Geräte im 20-Minuten-Takt zu arbeiten. Basierend auf der Tatsache, dass die durchschnittliche Transaktionslatenz im betrachteten Backend etwa 26296 ms beträgt, kann man daraus schließen, dass das Backend in der Lage ist, etwa 150 Transaktionen pro Stunde fehlerfrei zu verarbeiten. Wird dieser Wert überschritten, treten Fehler auf.

Dieses Problem entsteht, weil der Miner zu viele Transaktionen zur Bearbeitung erhält. In diesem Fall werden mehrere Transaktionen zu einem Blockchain-Block hinzugefügt. Somit wird mehr als ein Swarm-Hash in einem Block gespeichert. Das genutzte Backend kann mehrere Hashes innerhalb eines Blockes nicht unterscheiden. Dies führt dazu, dass der komplette Inhalt eines Blockes als ein Hash gelesen wird. Bei einem solchen Doppel-Hash können keine Daten in Swarm gefunden werden. Dies führt zu einem Fehler.

4.3 Speicher

Wie bereits in 2.3 ausgeführt, unterliegen IoT-Geräte Speicherbegrenzungen. Umso wichtiger ist es daher, zu verstehen, wie sich die Speicherkapazität in Bezug auf die untersuchte Problemgröße verhält. Im Rahmen des Experiments war die Speichergröße gut messbar. Die Messung wurde in der Weise unternommen, dass die Ethereum-Blockchaingröße und die Größe der Swarm-Knoten nach jeder Transaktion im Protokoll aufgezeichnet wurden. Das System der Datenspeicherung und Datensynchronisation sowohl im Ethereum, als

auch im Swarm stellt allerdings ein eigenständige Thema dar. Im Rahmen dieser Arbeit werden jedoch nur die Ergebnisse der Messung ohne weitere Analysen präsentiert.

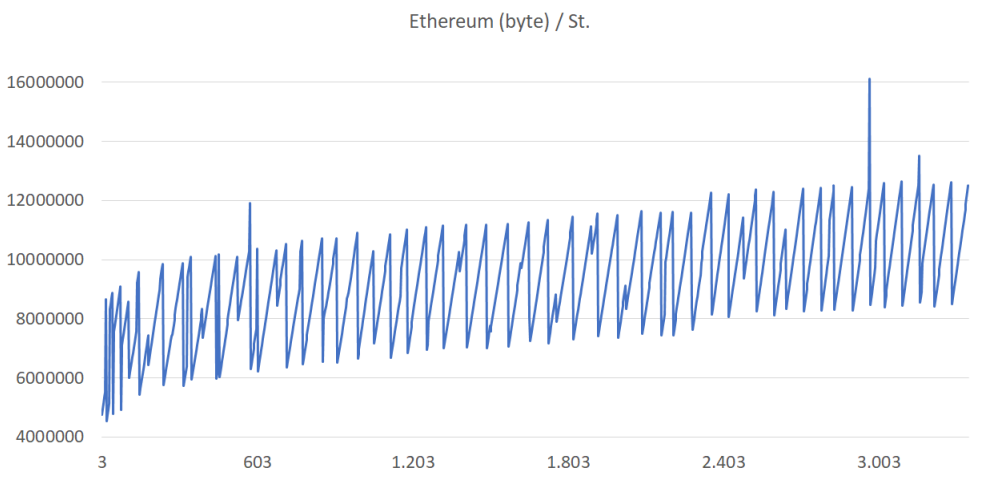

Abbildung 4.15: Ethereumgröße

Die Größe von Ethereum bei der Initialisierung betrug ca. 5,13 MB. Anschließend folgte eine lineare Steigerung im Verhältnis zu einer ebenfalls linear ansteigenden Zahl an Transaktionen (Abb. 4.15). Der Maximalwert betrug ca. 10,67 MB. Es stellte sich heraus, dass Ethereum 5,5 MB Speicher benötigte, um 3410 Swarm-Hashes in die Blockchain zu schreiben, dies entspricht 1,62 KB pro Eintrag.

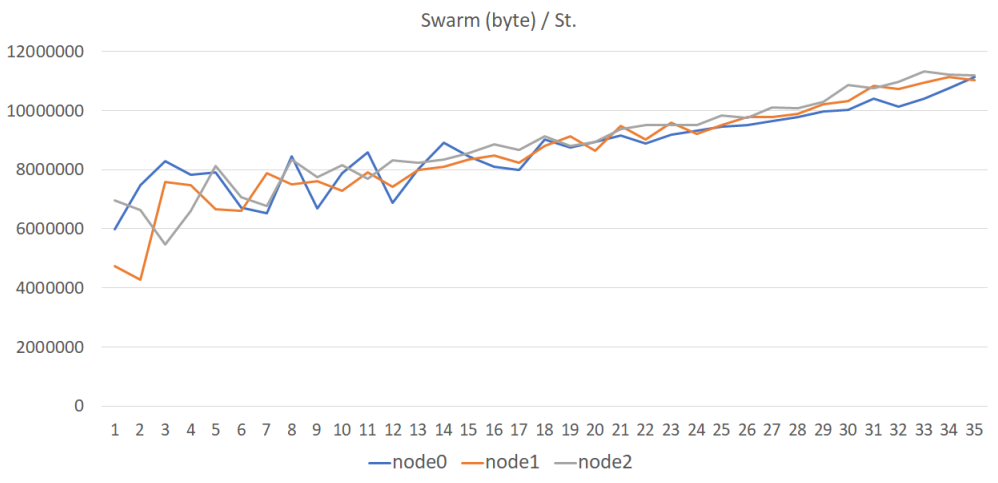

Abbildung 4.16: Swarmgröße

Die Größe der Swarm-Knoten während der Initialisierung variierte stark und reichte von

5 MB bis zu 7 MB. Bei der Verarbeitung der ersten tausend Transaktionen (innerhalb von 19 Stunden) konnten starke Schwankungen der Knotengröße beobachtet werden (Abb. 4.16). Ab der 20. Stunde konnte festgestellt werden, dass die Schwingungen geringer sind und die Knoten schneller miteinander synchronisierten. Am letzten Datenpunkt (36. Stunde) konnte Konvergenz der Swarm-Knoten auf den Wert von 11,11 MB verzeichnet werden.

5 Schlussfolgerungen

Das Ziel dieser Arbeit war es den Zusammenhang der Skalierbarkeit der Systemkosten von Blockchain-basierten IoT-Backend mit Ethereum und Swarm mit einer variierenden linear steigenden Problemgröße zu untersuchen. Dazu wurde eine Infrastruktur mit 2 IoT-Geräten und einem Server aufgebaut und mit einem Ethereum-Swarm-basierten Backend zusammengefasst. Um die Miningdauer zu verringern, wurde ein Mindestwert der Mining-Komplexität festgelegt und der Quellcode von Ethereum geändert, um eine Erhöhung des Komplexität-Parameters im Laufe des Experiments zu auszuschließen.

Mit Hilfe eines Testtools, das zum Zwecke dieser Arbeit entwickelt wurde, wurde innerhalb von 36 Stunden eine linear steigende Transaktionslast erzeugt. Während des Experiments wurden die Latenz, die Fehlerrate, sowie die Ethereumgröße und Swarmgröße gemessen und protokolliert. Die Ergebnisse aller untersuchten Geräte wurden präsentiert und analysiert.

Im Experiment konnte festgestellt werden, dass die Latenz des betrachteten Blockchain-basierten Backends linear ansteigt. Das Mining, das allein dadurch, dass es für den größten Teil der Transaktionslatenz verantwortlich ist, als erstes verdächtigt schien, hatte in Wirklichkeit, in Abhängigkeit vom Wachstum der Problemgröße, keine negativen Auswirkungen auf die Verzögerung. Stattdessen kam die lineare Latenzsteigerung durch die Smart Contract Funktion zustande, die für die Suche nach dem richtigen Zeitstempel zuständig war. Der Grund für diese Steigerung konnte darin gefunden werden, dass die Zeitstempel in einem üblichen Array gespeichert wurden. Die Suche nach dem notwendigen Element steigt linear mit der Anzahl der hinzugefügten Daten.

Bei einer durchschnittlichen Latenz von 26296 ms lag die Fehlerrate bei 1,94%. Eine detaillierte Analyse ergab, dass 100% der Fehler im Backend-Betrieb mit der Last von mehr als 150 Transaktionen pro Stunde aufgetreten sind. Aufgrund der großen Anzahl von Transaktionen begann Ethereum, mehrere Transaktionen zu einem Block zusammenzufügen, was von der vorgeschlagenen Backend-Architektur nicht vorgesehen war und zu Datenverlust führte.

Eine weitere Beobachtung bestand darin, dass die Ethereumgröße und die Swarmgröße linear mit der Anzahl an durchgeführten Transaktionen ansteigen. Bei der Verarbeitung der ersten tausend Transaktionen verzeichneten die Swarm-Knoten starke Schwankungen in der Speichergröße. Am Ende des Experiments betrug die Swarmgröße 11,11 MB. Das Ethereum wuchs in dieser Zeit um 5,5 MB auf 10,67 MB.

Die erläuterten Ergebnisse zeigen den Bedarf weiterer Analysen des Blockchain-basierten Backends, sowohl mit diesem hier verwendeten Technologie-Stack als auch mit weiteren, anderen Lösungen in diesem Bereich. Trotz alledem konnte diese Arbeit bestätigen, dass die Integration von IoT-Blockchain eine einsatzbereite und zuverlässige Backendlösung bietet. Besonders mit einer fortschreitenden Entwicklung in Richtung optimierter Konsens-Mechanismen für Blockchain, steht Blockchain zusammen mit IoT eine perspektivträchtige Zukunft bevor.

Literaturverzeichnis

[1] Hyperledger project website. Online erhältlich unter https://www.hyperledger.org/; abgerufen am 13.05.2019.

[2] Litecoin. Online erhältlich unter https://litecoin.org/; abgerufen am 15.05.2019.

[3] Multichain white paper. Online erhältlich unter https://www.multichain.com/white-paper/; abgerufen am 14.05.2019.

[4] Quorum whitepaper. Online erhältlich unter https://github.com/jpmorganchase/quorum-docs/blob/master/Quorum%20Whitepaper%20v0.1.pdf; abgerufen am 15.05.2019.

[5] Trusted IoT Alliance. Online erhältlich unter https://www.trusted-iot.org/; abgerufen am 29.04.2019.

[6] Ernst Baltensperger. *Monetary Economic Issues Today.* Orell Füssli Verlag, 2018.

[7] Tanja Kruse Brandão and Dr. Gerd Wolfram. *Digital Connection.* Springer Fachmedien Wiesbaden, 2018.

[8] M. Diaz, C. Martin, and B. Rubio. State-of-the-art, challenges, and open issues in the integration of internet of things and cloud computing. *Journal of Network and Computer Applications*, 67:99–117, Mai 2016.

[9] Swarm documentation. Online erhältlich unter https://swarm-guide.readthedocs.io/en/latest/introduction.html; abgerufen am 11.05.2019.

[10] Ethembedded. Online erhältlich unter http://ethembedded.com/; abgerufen am 01.05.2019.

[11] I. Eyal, A.E. Gencer, E.G. Sirer, and R. Van Renesse. Bitcoin-ng: a scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Designand Implementation (NSDI16)*, page 45–59, Santa Clara, CA, USA, 2016.

[12] Filament. Online erhältlich unter https://filament.com/; abgerufen am 02.05.2019.

[13] S. Gan. An iot simulator in ns3 and a key-based authentication architecture for iot devices using blockchain. Master's thesis, Indian Institute of Technology Kanpur, 2017.

[14] Gemalto. Almost half of companies still can't detect iot device breaches, reveals gemalto study. Online erhältlich unter https://bit.ly/2M6aeiN; abgerufen am 29.03.2019.

[15] A. Gupta, R. Christie, and R. Manjula. Scalability in internet of things: Features,techniques and research challenges. *International Journal of Computational Intelligence Research*, 13(7):1617–1627, 2017.

[16] J. Holthusen, S. Kufeld, and F. Glatz. Vorstellung derblockchain-technologie. Technical report, Deloitte Consulting GmbH, 2016.

[17] Swarm Home. Online erhältlich unter https://ethersphere.github.io/swarm-home/; abgerufen am 10.05.2019.

[18] S. Huh, S. Cho, and S. Kim. Managing iot devices using blockchain platform. In *2017 19th International Conference on Advanced Communication Technology (ICACT)*, pages 464–467, Feb 2017.

[19] Marco Iansiti and Karim R. Lakhani. The truth about blockchain. *Harvard Business Review*, January–February:118–127, 2017.

[20] Preethi Kasireddy. How does ethereum work, anyway? Online erhältlich unter https://medium.com/@preethikasireddy/how-does-ethereum-work-anyway-22d1df506369; abgerufen am 10.04.2019.

[21] M.A. Khan and K. Salah. Iot security: review, blockchain solutions, and open challenges. *Future Generation Computer Systems*, 82, Mai 2018.

[22] Jong-Hyouk Lee. Bidaas: Blockchain based id as a service. *IEEE Access*, 2017.

[23] F. Li, M. Voegler, M. Claessens, and S. Dustdar. Efficient and scalable iot service delivery on cloud. In *2013 IEEE Sixth International Conference on Cloud Computing*, pages 740–747, Santa Clara, CA, USA, 2013.

[24] M.Aazam and E.-N.Huh. Fogcomputing and smart gateway based communication for cloud of things. In *Proceedings of the 2nd International Conference on Future Internet of Things and Cloud*, pages 27–29, Barcelona,Spain, 2014. FiCloud-2014.

[25] Timothy McCallum. Running a quick ethereum private network for experimentation and testing, February 2018. Online erhältlich unter https://bit.ly/2Jerdeu; abgerufen am 16.06.2019.

[26] modum. Online erhältlich unter https://modum.io/; abgerufen am 02.05.2019.

[27] B. Montgomery. Future shock: Iot benefits beyond traffic and lighting energy optimization. *IEEE Consumer Electronics Magazine*, 4(4):98–100, 2015.

[28] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.

[29] Size of the Bitcoin blockchain from 2010 to 2019. Online erhältlich unter https://www.statista.com/statistics/647523/worldwide-bitcoin-blockchain-size/; abgerufen am 03.05.2019.

[30] Kazim Rifat Ozyilmaz and Arda Yurdakul. Designing a blockchain-based iot infrastructure with ethereum, swarm and lora. *CoRR*, abs/1809.07655, 2018.

[31] Koosha Paridel, Yves Vanrompay, Yolande Berbers, and Wolfgang De Meuter. Middleware for the internet of things, design goals and challenges. *Electronic Communications of the EASST*, 28, 2010.

[32] Joy Patra, Amitranjan Gantait, and Ayan Mukherjee. Governing your iot data: How to manage enterprise-wide iot initiatives, April 2018. Online erhältlich unter https://developer.ibm.com/articles/iot-governance-03/; abgerufen am 27.05.2019.

[33] G. Prisco. It to introduce smart locks linked to smart ethereum contracts decentralize the sharing economy, 2016. Online erhältlich unter https://bit.ly/2Jiz1P8; abgerufen am 02.05.2019.

[34] Kaa Project. Iot use cases. Online erhältlich unter https://www.kaaproject.org/iot-use-cases/; abgerufen am 28.03.2019.

[35] D. Puthal, N. Malik, S. P. Mohanty, E. Kougianos, and C. Yang. The blockchain as a decentralized security framework. *IEEE Consumer Electronics Magazine*, 7(2):18–21, 2018.

[36] Harald Pöttinger. Eintragen, Ändern und schützen der blockchain-datenstruktur. Online erhältlich unter http://haraldpoettinger.com/blockchain-datenstruktur-eintrag/; abgerufen am 16.05.2019.

[37] Raspnode. Online erhältlich unter http://raspnode.com/; abgerufen am 01.05.2019.

[38] Partha Pratim Ray. A survey of iot cloud platforms. *Future Computing and Informatics Journal*, 1(1-2):35–46, December 2016.

[39] A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz. On blockchain and its integration with iot. challenges and opportunities. *Future Generation Computer Systems*, 88:173–190, November 2018.

[40] J. Rivera and R. van der Meulen. Das internet der dinge: So verändert die nächste dimension des internet die welt. Technical report, Cisco Internet Business Solutions Group (IBSG), April 2011.

[41] J. Rivera and R. van der Meulen. Forecast alert: Internet of things — endpoints and associated services. Technical report, Gartner, 2016.

[42] R. Roman, J. Zhou, and J. Lopez. On the features and challenges of security and privacy in distributed internet of things. *Computer Networks*, 57(10):2266–2279, Mai 2013.

[43] Ant Router R1-LTC The WiFi router that mines Litecoin. Online erhältlich unter https://shop.bitmain.com/antrouter_r1_ltc_wireless_router_and_asic_litecoin_miner.html; abgerufen am 02.05.2019.

[44] M. Samaniego and R. Deters. Blockchain as a service for iot. In *2016 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 433–436, Dec 2016.

[45] M. Samaniego and R. Deters. Hosting virtual iot resources on edge-hosts with blockchain. In *Computer and Information Technology (CIT)*, pages 116–119, IEEE International Conference on, Yanuca Island, Fiji, 2016. IEEE.

[46] J. Scherk and G. Pöchhacker-Tröscher. Die blockchain–technologie-feld und wirtschaftliche anwendungsbereiche. Technical report, BMVIT - Bereich Innovation, Mai 2017.

[47] Ethereum Chain Data Size Growth (FAST Sync). Online erhältlich unter https://etherscan.io/chart2/chaindatasizefast; abgerufen am 03.05.2019.

[48] Maarten van Steen and Andrew S. Tanenbaum. *Distributed Systems*. Maarten van Steen, 3. edition, 2017.

[49] P. Veena, S. Panikkar, S. Nair, and P. Brody. Empowering the edge-practical insights on a decentralized internet of things. Technical report, IBM Institute for Business Value, 2015.

[50] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In Jan Camenisch and Doğan Kesdoğan, editors, *Open Problems in Network Security*, pages 112–125, Cham, 2016. Springer International Publishing.

[51] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper, 2014.

[52] A. Zamyatin, K. Wolter, S. Werner, P. G. Harrison, C. E. A. Mulligan, and W. J. Knottenbelt. Swimming with fishes and sharks: Beneath the surface of queue-based ethereum mining pools. In *2017 IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 99–109, Sep. 2017.

A Appendix

A.1 genesis.json

```
{
  "config": {
      "chainId": 42,
      "homesteadBlock": 0,
      "eip155Block": 0,
      "eip158Block": 0
  },
  "nonce": "0x0000000000000042",
  "timestamp": "0x0",
  "parentHash": "0x0000000000000000000000000000000000000000000000000",
  "gasLimit": "0x8000000",
  "difficulty": "0x1",
  "mixhash": "0x0000000000000000000000000000000000000000000000000",
  "coinbase": "0x3333333333333333333333333333333333333333",
  "alloc": {
      "0xe83899a147f27d166a7c71c86f6848c25dbc3628": {
          "balance": "2000000000000000000000000000"
      },
      "0x21fa004490ad6048152d2a8164c18e01a8e1a165": {
          "balance": "2000000000000000000000000000"
      },
      "0x2cf16ed097433139ac4da9f571afde22de6ceb57": {
          "balance": "2000000000000000000000000000"
      }
  }
}
```

A.2 default.json

```
{
  "deviceId": "node0",
  "ethAddr" : "http://localhost:8001",
  "ethAccount" : "0x29bd1477a6879226fd489472f2db498276e94ca3",
  "password" : "haw",
  "contractAddr" : "0x1aaa19379c740e5e466f45053ebd2a3df8ff67a5"
     ,
  "swarmAddr" : "http://localhost:8500/bzz:/",
  "gethFolder" : "/home/arty/node/geth",
  "swarmFolder" : "/home/arty/node/swarm"
}
```

A.3 vendor.sol

```
pragma solidity ^0.4.6;

contract vendor {
    // contract owner
    address private creator;
    // IoT device data
    struct device_data {
        // link for detecting devices
        uint index;
        // blockchain timestamps stored swarm hashes
        uint[] timestamps;
        // map timestamp values to swarm file hashes
        mapping(uint => string) filehashes;
    }
    // map device id's to their data (one data per id)
    mapping(address => device_data) private device_logs;
    // keep a separate device id array of all received id's
    address[] private device_index;
    // event to log action
    event log_action (address indexed device_id, uint index,
     uint timestamp, string filehash);
    // check if device is seen before?
    function is_device_present (address device_id)
    public constant returns (bool result) {
        // return false if no device present yet!
        if(device_index.length == 0) return false;
        // return true if device exists
        return (device_index[device_logs[device_id].index]
        == device_id);
    }
    // helper function for string concatenation
    // http://cryptodir.blogspot.com.tr/2016/03/solidity-concat
       -string.html
    function filehash_concat (string h1, string h2) internal
```

```
        constant returns (string concat) {
34          bytes memory b1 = bytes(h1);
35          bytes memory b2 = bytes(h2);
36          string memory sm = new string(b1.length + 1 + b2.length
                );
37          bytes memory bm = bytes(sm);
38          uint i = 0;
39          uint k = 0;
40          for (i = 0; i < b1.length; i++) bm[k++] = b1[i];
41          bm[k++] = ',';
42          for (i = 0; i < b2.length; i++) bm[k++] = b2[i];
43          return string(sm);
44      }
45      // push specific device data handle into the chain
46      function set_device_data (address device_id, string
            filehash) public returns (uint index, uint timestamp) {
47          uint ts;
48          if(is_device_present(device_id)) {
49              // device already exists
50              ts = now;
51              if( device_logs[device_id].timestamps.length == 0
                    ||
52                  device_logs[device_id].timestamps[device_logs[
                        device_id].timestamps.length-1] < ts) {
53                  // first insertion or insertion made (at least)
                        in last block
54                  device_logs[device_id].timestamps.push(ts);
55                  device_logs[device_id].filehashes[ts] =
                        filehash;
56              } else if(device_logs[device_id].timestamps[
                    device_logs[device_id].timestamps.length-1] ==
                    ts) {
57                  // data inserted before, lets concat data for
                        that certain timestamp
58                  device_logs[device_id].filehashes[ts] =
                        filehash_concat(device_logs[device_id].
```

```
                    filehashes[ts], filehash);
59              } else {
60                  // doc, this is heavy!
61                  assert(false);
62              }
63              // trigger event
64              log_action(device_id, device_logs[device_id].index,
                    ts, filehash);
65              return(device_logs[device_id].index, ts);
66          } else {
67              // device received first time
68              ts = now;
69              device_logs[device_id].timestamps.push(ts);
70              device_logs[device_id].filehashes[ts] = filehash;
71              device_logs[device_id].index = device_index.push(
                    device_id)-1;
72              // trigger event
73              log_action(device_id, device_index.length-1, ts,
                    filehash);
74              return(device_index.length-1, ts);
75          }
76      }
77      // get received data at a certain timestamp for a specific
            device
78      function get_device_data (address device_id, uint timestamp
            ) public constant returns (string hash) {
79          if(!is_device_present(device_id)) revert();
80          return device_logs[device_id].filehashes[timestamp];
81      }
82      // get all data timestamps for a specific device
83      // TODO create a timestamp getter() with start-end interval
84      function get_device_timestamps (address device_id) public
            constant returns (uint[] timestamp) {
85          if(!is_device_present(device_id)) revert();
86          return device_logs[device_id].timestamps;
87      }
```

```
88      // get total device count
89      function get_device_count() public constant returns (uint
            count) {
90          return device_index.length;
91      }
92      // get device address from index
93      function get_device_at_index (uint index) public constant
            returns (address device_address) {
94          return device_index[index];
95      }
96      // constructor
97      function vendor() {
98          creator = msg.sender;
99      }
100     // kills contract and sends remaining funds back to creator
101     function kill() {
102         if (msg.sender == creator) {
103             selfdestruct(creator);
104         }
105     }
106 }
```

Listing A.1: Datenstruktur

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Kostenanalyse einer Blockchain-basierten IoT-Infrastruktur mit Ethereum und Swarm

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original