h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Hochschule Darmstadt
- Fachbereich Informatik -

Permissioned Blockchains für B2B

Prototypische Implementierung eines dezentralisierten Wartungsmarktes

Abschlussarbeit zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.)

vorgelegt von

Eric Nagel
Matrikelnummer 740693

Referent:	Prof. Dr. Andreas Müller
Korreferent:	Björn Bär

Eigenständigkeitserklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, den 21. Februar 2018

Eric Nagel

Zusammenfassung

Traditionelle B2B-Anwendungen, mit multiplen Unternehmen als Teilnehmer, bringen Probleme bezüglich der Datenhaltung mit sich. Eine Option ist, dass jedes Unternehmen Daten bei sich selbst speichert. Dies führt jedoch zu zusätzlichen Aufwand, da Schnittstellen eingerichtet werden müssen, um Geschäftspartnern Zugriff auf relevante Daten zu gewähren. Eine weitere Option ist die Speicherung bei einer zentralen, verwaltenden Instanz. Jeder Geschäftspartner müsste dieser jedoch vertrauen, was die Anwendung unattraktiver macht. Eine Lösung für diese Probleme könnte die Blockchain-Technologie darstellen. Mit ihr ist eine verteilte Datenspeicherung unter nicht vertrauenswürdigen Teilnehmern möglich. Dabei stellt die Blockchain sicher, dass die Daten bei allen Geschäftspartnern synchron sind und nicht manipuliert oder gelöscht werden können. Die Technologie bringt jedoch auch Schwierigkeiten mit sich, welche für B2B-Anwendungen unerwünscht sind. In Public Blockchains wie Bitcoin und Ethereum äußert sich dies anhand der Skalierbarkeit, Performance und Sicherheit. Permissioned Blockchains, wie Hyperledger Fabric, können diese Probleme zu Teilen lösen. Nichtsdestotrotz sind darauf basierende Anwendungen auf eine bestimmte Performance oder Nutzerzahl limitiert. Auf Grundlage dieser Erkenntnisse wurde ein Prototyp eines automatisierten dezentralen Wartungsmarktes mit Hyperledger Fabric als Proof-of-Concept entwickelt. In diesem können IoT-Geräte Wartungsbedarf erkennen und Wartungsanbieter darauf reagieren. Beliebige Teilnehmer können an dem Markt teilnehmen, ohne dass Datenmanipulation durch eine Partei zu befürchten ist. Wartungen werden verfolgbar und unveränderbar dokumentiert, ohne dass Vertrauen zwischen den Parteien nötig ist.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Abkürzungsverzeichnis

PoW Proof-of-Work

TPS Transaktionen pro Sekunde

BFT Byzantine Fault Tolerance

PBFT Practical Byzantine Fault Tolerance

BND Business Network Definition

BNA Business Network Archive

Kapitel 1

Einführung und Motivation

Klassische B2B-Anwendungen bringen Probleme hinsichtlich der Datenhaltung mit sich. Eigene Daten können bei jeden Geschäftspartner selbst gespeichert werden, was jedoch den Zugriff auf diese, aufgrund von aufwendig einzurichtenden Schnittstellen und uneinheitlichen Datenformaten, erschwert. Eine weitere Möglichkeit ist die Speicherung bei einem zentralen Unternehmen. Dieses hätte jedoch die Kontrolle über die Daten, womit alle anderen Parteien diesem vertrauen müssten. Diese Faktoren machen B2B-Anwendungen für die Teilnehmer unattraktiv und erschweren die Entwicklung [1][2].

Um diese Probleme zu lösen wird ein Prototyp einer dezentralen B2B-Applikation, basierend auf der Blockchain-Technologie, entwickelt. Sie erlaubt es dezentrale Systeme aufzubauen, in welchen sich die Parteien nicht vertrauen. Alle Daten würden bei jedem Teilnehmer des Blockchain-Netzwerks (im Folgenden nur noch Netzwerk genannt) gespeichert werden. Trotzdem sind diese nicht lösch- oder manipulierbar, alle Transaktionen sind lückenlos nachvollziehbar und es besteht ein gemeinsamer Konsens über den Datenbestand [3].

Bekannte Blockchain-Implementationen, wie Bitcoin oder Ethereum, bringen jedoch Probleme für den B2B-Bereich mit sich. So sind alle Daten öffentlich einsehbar, der Transaktionsdurchsatz ist gering und die Konsensmechanismen sind unter bestimmten Umständen unsicher und resultieren in hohen Energieverbrauch [4][5][6].

Ziel dieser Arbeit ist es, die Probleme der Blockchain-Technologie für den B2B-Bereich zu analysieren und basierend auf den Ergebnissen eine dezentrale B2B-Anwendung als Proof-of-Concept zu entwickeln. Dazu werden zunächst die grundlegenden Konzepte der Blockchain-Technologie erläutert, um ein besseres Verständnis für die Vor- und Nachteile dieser zu erhalten. Anschließend werden die Probleme für B2B-Anwendungen anhand der Anforderungen an diese genauer betrachtet und analysiert. Daraufhin erfolgt die Beschreibung der Anwendungsentwicklung. Zuletzt wird ein Fazit zur Lösung der Probleme und des entwickelten Systems gezogen.

Kapitel 2

Grundlagen

Zum Verstehen der Diskussion der Probleme der Blockchain-Technologie ist es notwendig, die grundlegenden Konzepte dieser zu verstehen. Weiterhin müssen allgemeine Anforderungen an B2B-Anwendungen erfasst werden, anhand welchen die Diskussion erfolgt.

2.1 Blockchain

2.1.1 Funktionsweise

Die Funktionsweise der Blockchain wird in dieser Arbeit hauptsächlich am Beispiel von Bitcoin erklärt. Als erste Blockchain-Anwendung [7] und aufgrund der überschaubaren Komplexität liefert es die Grundlage für die Funktionsweise der Technologie. Andere Implementationen, wie Ethereum oder Ripple, funktionieren nach dem gleichen Prinzip.

2.1.1.1 Allgemein

Wenn der Begriff "Die Blockchain" auftaucht, ist damit meistens die Blockchain-Technologie gemeint. Es gibt nicht nur eine global bestehende Blockchain und auch nicht nur eine Implementation der Technologie, was beispielsweise anhand von Bitcoin und Ethereum ersichtlich ist.

Allgemein kann die Blockchain als Datenstruktur bezeichnet werden, welche verteilt, nicht löschbar und unmanipulierbar gespeichert werden kann. Weiterhin verifizieren jegliche Teilnehmer am Netzwerk ausgeführte Transaktionen, wodurch ein gemeinsamer Konsens über den Datenbestand besteht [3].

In einer Blockchain werden Transaktionen in Blöcken gespeichert. Dabei handelt es sich um Operationen, welche Daten erstellen, verändern, oder löschen. Aus diesen lässt sich letztendlich der aktuelle Datenbestand ermitteln. So erfolgt beispielsweise bei Bitcoin keine Speicherung des aktuellen Guthabens der Teilnehmer. Es wird nur aus allen bestehenden Transaktionen berechnet [8, S. 85]. Die Daten, welche letztendlich bestehen, können beispielsweise Geldtransferinformationen (Bitcoin), Smart Contracts (Ethereum, selbst ausführende Verträge mit selbst

erstellter Programmlogik, siehe 2.1.3), simple Dokumente oder Informationen sein [6][5][9]. Die Blöcke setzen sich aus den Transaktionen sowie den Block Header zusammen, welcher verschiedene Metadaten, wie zum Beispiel den kombinierten Hash[1] aller Transaktionen, enthält [8, S. 160-161].

Die Blöcke sind miteinander verkettet. Jeder Block Header enthält den Hash des vorherigen Block Headers (siehe Abb. 2.1). Dies ist ein wichtiges Feature zum Schutz der Blockchain vor Angriffen. Wenn ein Angreifer die Transaktionen eines Blocks zu seinen Gunsten verändern würde, würde sich der Hash des Block Headers ändern. Dieser müsste dann im darauffolgenden Block Header stehen, wodurch sich allerdings auch der Hash dieses Blocks ändert. Letztendlich müssten alle nachfolgenden Blöcke manipuliert werden, um eine gültige Blockchain zu erhalten [5]. Diese Manipulation wird durch verschiedene Verfahren erschwert, welche genauer in den Kapiteln 2.1.1.1 und 2.1.1.3 erklärt werden.

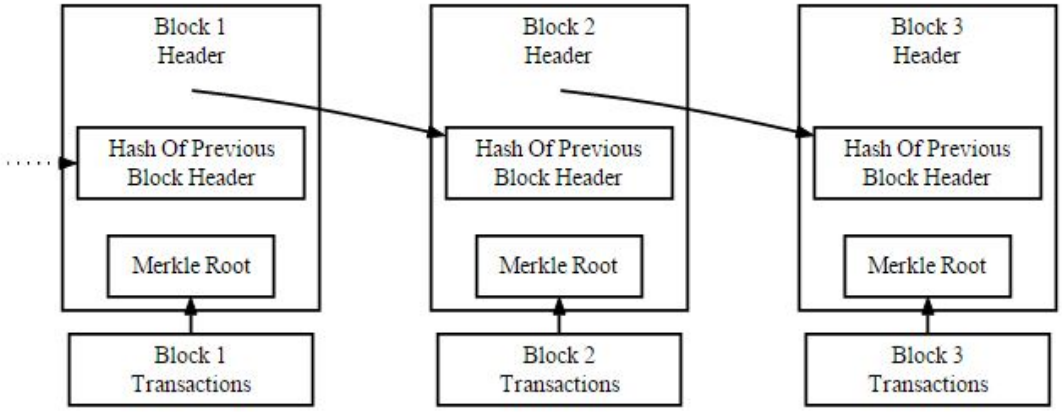

Abbildung 2.1: Verkettung von Blöcken durch Block Header Hashes [11].

Die Blockchain ist verteilt gespeichert. Jeder Teilnehmer hat die Möglichkeit Sie auf seinen Rechner zu speichern. Somit besteht keine zentrale Instanz, welche die Kontrolle über die Daten hat. Weiterhin gibt es keinen Single Point of Failure[2] [12].

2.1.1.2 Konsensmechanismen

Aufgrund der verteilten Datenhaltung muss es Verfahren geben, um die Daten synchron und auf einen Stand, auf welchen sich alle Teilnehmer geeinigt haben, zu halten. Dazu gibt es die sogenannten Konsensmechanismen, welche gleichzeitig die Unmanipulierbarkeit der Daten sicherstellen. Bevor diese erklärt werden können, muss zunächst genauer auf die Funktion des Netzwerks eingegangen werden.

[1] Hash: Ergebnis einer Operation, welche Daten in eine Zeichenfolge fester Länge umwandelt [10, S. vii].

[2] Single Point of Failure: Komponente eines Systems, dessen Ausfall den Ausfall des gesamten Systems bewirkt [12].

Funktion des Netzwerks Wenn ein Teilnehmer eine Transaktion ausführt, wird diese, vorausgesetzt dass sie valide ist (genauer im nächsten Absatz erklärt), an alle Nodes[3] im Netzwerk weitergeleitet und im Transaktionspool aufgenommen. Dieser enthält alle noch nicht in Blöcken vorkommenden Transaktionen. Diese werden in einen neuen Block aufgenommen während jede Node mit der Erstellung von diesem beschäftigt ist. Dies wird durch verschiedene Konsensmechanismen realisiert. Bei Bitcoin und Ethereum findet der Proof-of-Work (PoW) Anwendung (weiter unten genauer unter erklärt). Sobald eine Node einen Block erstellt, wird dieser im Netzwerk verteilt. Jede Node hängt ihn an ihre lokale Blockchain an und beginnt mit der Erstellung des nächsten Blocks [8, S. 200 ff.].

Damit eine Transaktion valide ist, muss sie bestimmte Voraussetzungen erfüllen. So muss sie unter anderem mit dem Private Key des Senders signiert sein. Mittels seines Public Keys kann überprüft werden, ob wirklich er der Sender der Nachricht ist und ob die Transaktion manipuliert wurde. Dieses Verfahren wird auch in der Abbildung 2.2 visualisiert. Das Signieren trägt zur Sicherheit der Blockchain bei, da ein Angreifer somit keine Transaktionen eines Nutzers manipulieren oder in seinem Namen ausführen kann. In Bitcoin ist eine weitere Kondition, dass der Transaktionsersteller die zu sendenden Bitcoins besitzt [8, S. 18]. In Systemen wie Ethereum und Hyperledger Fabric, in welchen eigene Programmlogik abgebildet werden kann, können weitere Konditionen festgelegt werden. So muss beispielsweise ein Teilnehmer die nötigen Rechte haben um eine Transaktion auszuführen [13].

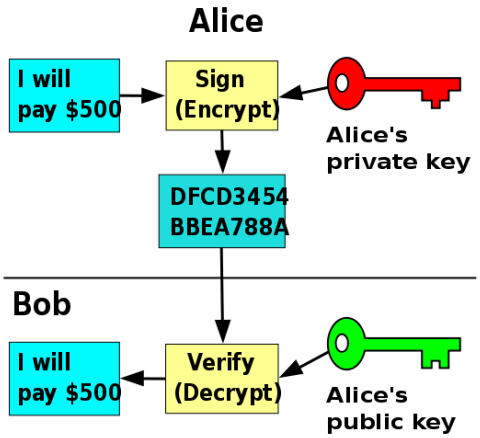

Abbildung 2.2: Signieren und Verifizieren von Nachrichten. Der Sender signiert die Nachricht mit seinem Private Key und der Empfänger kann diese mit den Public Key des Senders verifizieren [14].

Proof-of-Work Der PoW ist nur einer der zur Verfügung stehenden Konsensmechanismen (siehe Kapitel 4.2). Er bedarf jedoch genauerer Erklärung, da er in aktuellen Blockchain-Implementationen vorwiegend genutzt wird. Der PoW ist eine Art Rätsel, welches mit der Rechenleistung von Nodes gelöst werden muss, um einen Block zu erschaffen (das sogenannte

[3]Node: Teilnehmer eines Blockchain-Netzwerks, welche die Blockchain speichern. Auch Peers genannt.

Mining). Genauer gesagt, muss für einen Block ein Hash gefunden werden, welcher einen bestimmten Wert unterschreitet. Desto kleiner dieser ist, desto höher ist die Schwierigkeit. Für Sicherheitszwecke ist es nötig, dass die Blockerstellung eine gewisse Zeit benötigt. Um wachsender Rechenleistung und Teilnehmerzahl entgegenzuwirken und somit diese Zeit zu garantieren, kann die Schwierigkeit angepasst werden. Dies wird genauer im Kapitel 4.1 erläutert. Um unterschiedliche Hashwerte für gleiche Blöcke zu erhalten, gibt es im Block Header eine Nonce[4], welche verändert wird [5]. Alle Nodes im Bitcoin-Netzwerk benötigen im Durchschnitt 10 Minuten, um einen PoW zu erbringen [8, S. 173], bei einer Hash Rate[5] von ca. 13.000.000 TH/s (Terrahashes pro Sekunde) [17]. Bei Ethereum beträgt die Zeit ungefähr 15 Sekunden [18], bei einer Hash Rate von ca. 150 TH/s [17]. Spezielle Hardware für das Mining erreicht eine Hashrate von bis zu 13,5 TH/s [19]. Damit die Nodes eine Motivation haben, Rechenleistung für das Erstellen von Blöcken zu nutzen, erhalten sie bei Erbringung des PoW eine Belohnung sowie die Transaktionsgebühren in Form von Kryptowährung. In Verbindung mit den Mining sind auch Mining Pools zu erwähnen. Dabei handelt es sich um einen Zusammenschluss von Minern, welche gemeinsam probieren den PoW zu erbringen. Gelingt dies, wird die Belohnung zwischen allen Teilnehmern im Pool geteilt. [5] [6].

Forking Um vollständig zu verstehen, wie der PoW funktioniert, muss das Forking erklärt werden. Wenn eine Node einen PoW erbringt, also einen Block erstellt, wird dieser an alle anderen Nodes weitergeleitet. Im Bitcoin-Netzwerk dauert es bei einer maximalen Blockgröße von 1 MB [20], zwischen 6 und 20 Sekunden, bis ein Block mindestens 90 % aller Nodes erreicht hat [21]. Dies stimmt auch mit dem Paper von Decker und Wattenhofer überein, wo eine durchschnittliche Zeit von 12,6 Sekunden angegeben wird, bis ein Block 95 % aller Nodes erreicht [22]. In dieser Zeit kann es vorkommen, dass eine weitere Node einen Block erstellt. Auch dieser wird im Netzwerk verteilt, wodurch zwei Versionen der Blockchain existieren: Eine endet mit Block A und die andere mit Block B. Dies ist der sogenannte Fork. Das Netzwerk muss sich nun darauf einigen, welche der beiden Versionen beibehalten werden soll. Deshalb gilt: Die Blockchain, in welche mehr Arbeit eingeflossen ist, ist die gültige. Im Falle von Bitcoin wäre dies die längere Blockchain. Die Nodes probieren an den zuerst erhaltenen Block (A oder B) einen neuen anzuhängen. Gelingt dies, ist eine der beiden Blockchains länger als die andere. Diese wird dann von allen Nodes als die richtige akzeptiert. Dieser Vorgang wird auch in den Abbildungen 2.3 bis 2.6 dargestellt. Theoretisch ist es möglich, dass ein Fork über mehrere Blöcke besteht. Die Wahrscheinlichkeit dafür ist jedoch gering, da mehrmals nacheinander mindestens zwei Nodes zur ungefähr gleichen Zeit einen Block erstellen müssen. Auch zu erwähnen ist, dass in einem Fork-Branch weitere Forks entstehen können. Diese Forks sind der Grund, warum Transaktionen erst als bestätigt gelten, sobald sie in einem Block stehen, welcher eine gewisse Anzahl an Nachfolgern hat. Denn erst dann ist die Sicherheit gegeben, dass die Transaktion nicht in einem Branch vorhanden ist, welcher eventuell verworfen wird [8, S. 211 ff.]. Wie genau

[4]Nonce: Eine Nummer, welche genau einmal für einen bestimmten Zweck genutzt wird [15].

[5]Hash Rate: Anzahl der in einer Zeiteinheit berechneten Hashwerte [16].

der PoW das Netzwerk absichert, wird im Kapitel 2.1.1.3 erklärt.

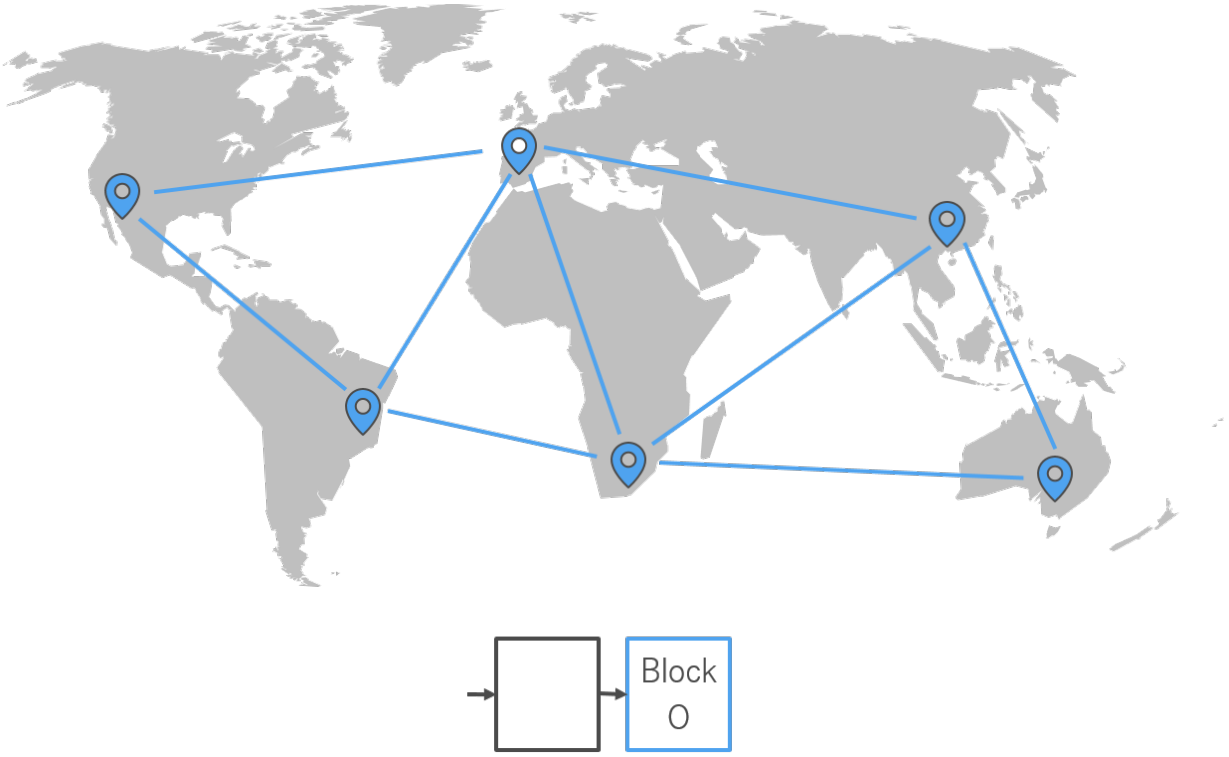

Abbildung 2.3: Fork-Visualisierung - Vor dem Fork besitzen alle Nodes Block O als letzten Block [8, S. 200 ff.].

Neben dem PoW gibt es noch weitere Konsensmechanismen, wie Proof-of-Stake, Proof-of-Authority oder Practical Byzantine Fault Tolerance [23], [24] . Diese werden im Kapitel 4.2 genauer beschrieben und analysiert.

2.1.1.3 Nichtangreifbarkeit und Unveränderlichkeit

Viele Faktoren tragen zur Nichtangreifbarkeit und Unveränderlichkeit (im weiteren Verlauf nur noch als Sicherheit bezeichnet) der Blockchain bei. Da alle Nodes die ausgeführten Transaktionen auf Validität prüfen, können diese nicht ohne Berechtigung, im Namen einer anderen Identität oder mit unzureichenden Konditionen ausgeführt werden. Der wichtigste Faktor ist jedoch der genutzte Konsensmechanismus in Verbindung mit den verketteten Blöcken. Durch ihn wird sichergestellt, dass bestehende Daten nicht gelöscht oder manipuliert werden können.

Ein Beispiel dafür kann am PoW gezeigt werden. Ein Angreifer probiert eine Transaktion aus einen bestehenden Block zu entfernen. Dazu würde er die Transaktion bei seiner lokalen Blockchain entfernen. Nun ist jedoch der Hash des Blockes sowie der Block selbst nicht valide und würde von keiner Node akzeptiert werden. Der Angreifer muss also erneut einen PoW für den manipulierten Block erbringen. Dies wäre für eine Einzelperson jedoch zeitaufwändig. Wenn der manipulierte Block nun noch Nachfolger hat, muss aufgrund des neuen Hashes auch für diese der PoW erbracht werden. Hinzu kommt, dass die Blockchain des Angreifers erst

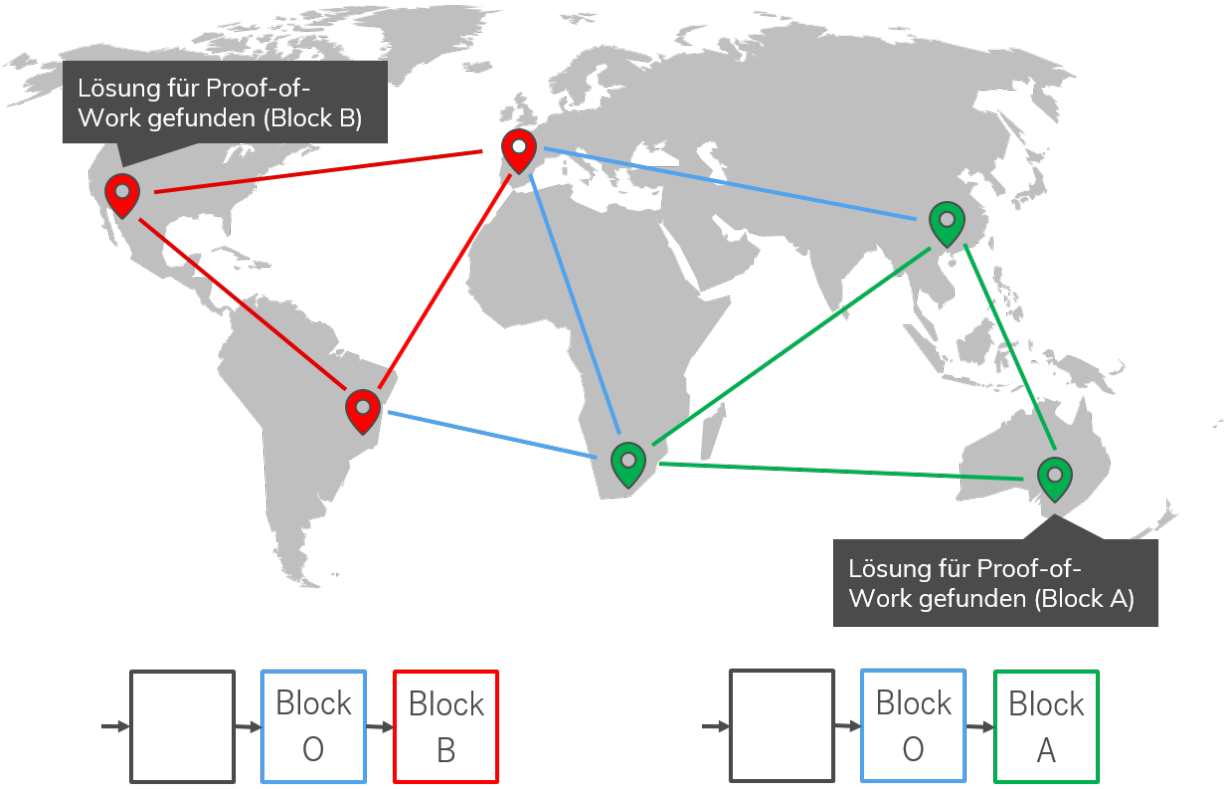

Abbildung 2.4: Fork-Visualisierung - zwei Nodes finden zur ungefähr gleichen Zeit einen Block und verbreiten ihn im Netzwerk, wodurch zwei Versionen der Blockchain bestehen [8, S. 200 ff.].

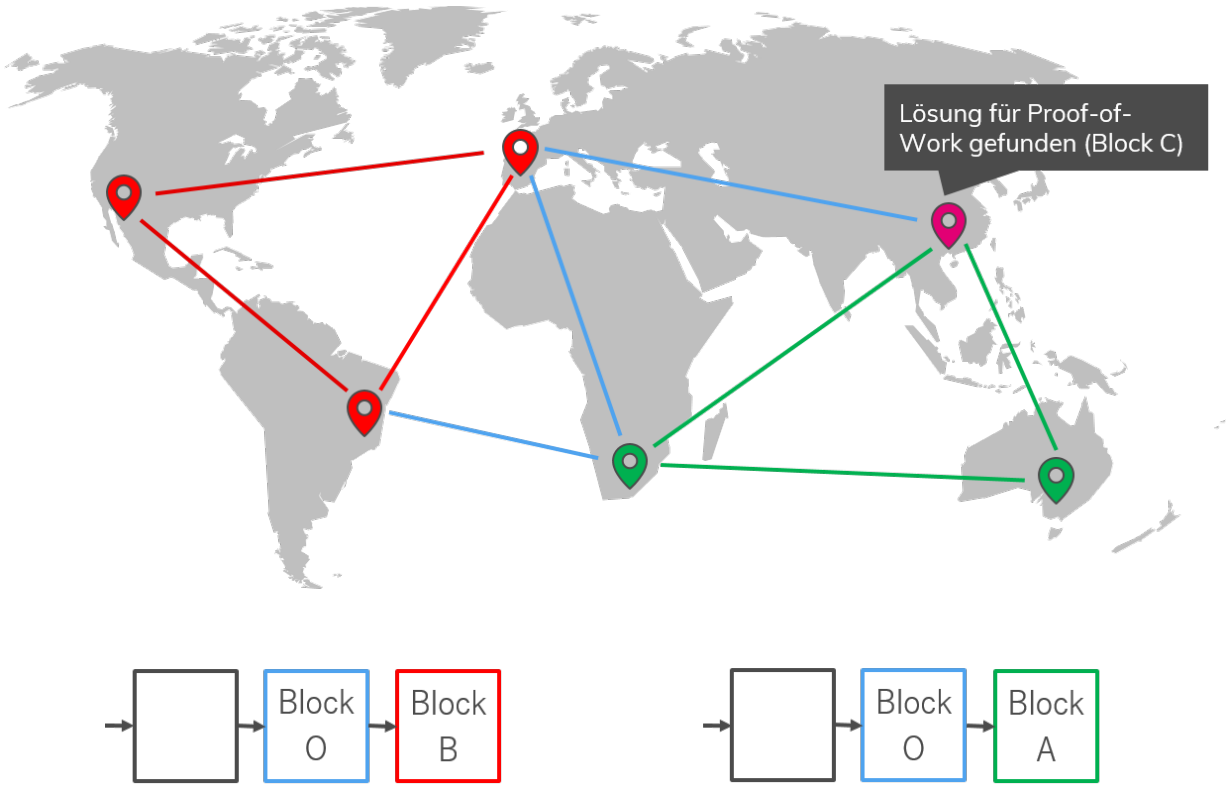

Abbildung 2.5: Fork-Visualisierung - Eine Node, welche Block A zuerst erhalten hat, hängt daran einen neuen Block C an [8, S. 200 ff.].

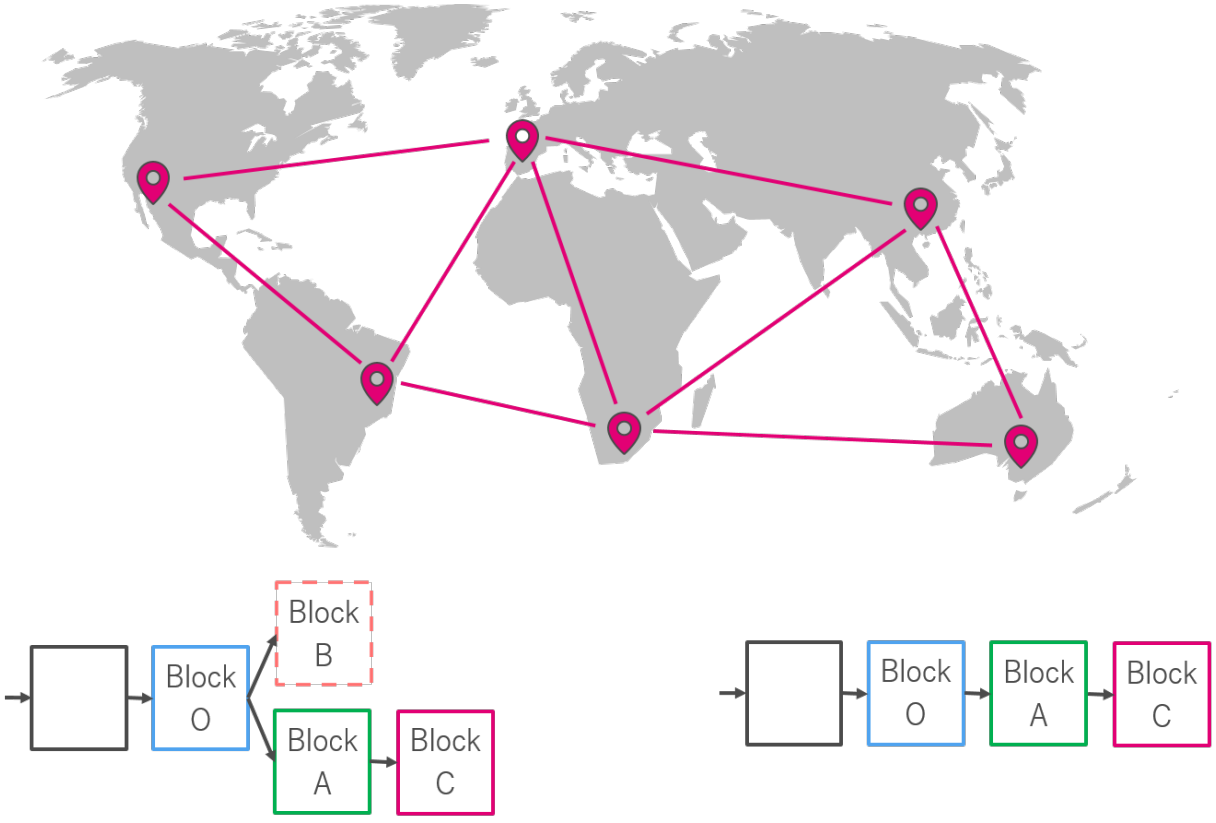

Abbildung 2.6: Fork-Visualisierung - Block C verbreitet sich im Netzwerk, rote Nodes sehen zwei Blockchains und akzeptieren die längere [8, S. 200 ff.].

von allen Nodes akzeptiert wird, wenn sie länger ist als die aktuell akzeptierte. Er müsste also schneller als das gesamte Bitcoin-Netzwerk Blöcke erschaffen können. Dies ist nur möglich, wenn er 51 % der Rechenleistung des Netzwerks besitzt. Deshalb wird dieser Angriff auch 51 %-Angriff genannt [10, S. 83] [6].

An dieser Stelle sollte erwähnt werden, dass, auch wenn ein 51%-Angriff erfolgt, die Angriffsmöglichkeiten beschränkt sind. Der Angreifer kann keine invaliden Transaktionen sowie Blöcke erstellen. Ihm ist es möglich DoS-Angriffe auszuführen, indem er verhindert, dass bestimmte Transaktionen in Blöcken aufgenommen werden. Ebenso kann er die Historie der Daten verändern, indem er eine Transaktion aus einem Block entfernt. Es ist jedoch zu bedenken, dass die Transaktion dabei zurück in den Transaktionspool geht. Er muss also jeden weiteren Block erstellen, um zu verhindern, dass die Transaktion nicht erneut in einen Block aufgenommen wird oder dafür sorgen, dass sie ungültig wird. Im Falle von Kryptowährungen wird ein solcher Angriff Double-Spending-Angriff genannt: Ein Angreifer sendet beispielsweise Bitcoins an einen Händler. Dieser wartet auf die Bestätigung der Transaktion in einen Block sowie auf nachfolgende Blöcke. So stellt er sicher, dass die Transaktion nicht in einem eventuell verworfenen Fork-Branch steht. Erst dann versendet er die Ware. Anschließend ersetzt der Angreifer die Transaktion durch eine Zahlung an sich selbst und erstellt die längere Blockchain. Somit wird die Transaktion an den Händler ungültig, wodurch er letztendlich doch kein Geld erhalten hat [6]. Auch zu bedenken ist, dass ein Nutzer mit 51 % der Rechenleistung wenig Motivation hat Angriffe auszuführen, da er für jeden erstellten Block Kryptowährung als Belohnung erhält. Ebenfalls würde der Wert dieser sinken, wenn Angriffe auf die Blockchain entdeckt werden. Deshalb besteht für die sogenannten Miner eine Motivation, ehrlich zu arbeiten [8, S.196 ff.].

Der eben genannte Double-Spending-Angriff kann auf verschiedene Arten durchgeführt werden. So könnte ein Angreifer eine Ware mit Bitcoins bezahlen und diese sofort erhalten. Bevor seine Transaktion jedoch in einen Block besteht, transferiert er die gleichen Bitcoins an sein anderes Konto. Wenn diese beiden Transaktionen letztendlich in einen Block aufgenommen werden sollen, wird eine der Transaktionen ungültig sein, wodurch der Verkäufer eventuell keine Zahlung erhält [8, S. 211 ff.].

2.1.2 Blockchaintypen

Anhand der zugelassenen Teilnehmer an einer Blockchain, ergeben sich 3 Typen. Bisher wurden nur Public Blockchain-Anwendungen, wie Bitcoin und Ethereum, erwähnt. In diesen gibt es keine Teilnehmerbeschränkungen, jeder kann am Netzwerk teilnehmen und die Blockchain öffentlich einsehen. Anders ist dies bei Permissioned (oder auch Consortium [25]) und Private Blockchains. Die beiden Begriffe werden in einigen wissenschaftlichen Arbeiten gleichgesetzt (siehe [4], [**PongnumkulPerformanceAnalysisPrivate2017**], [26]). Hier folgt jedoch eine Unterscheidung. Dabei ist eine Private Blockchain eine Blockchain, welche nur von einem Nutzer verwendet wird. Da eine solche Anwendung keinen Sinn ergibt, da keine Vorteile der Blockchain genutzt werden können, wird darauf nicht genauer eingegangen. Interessanter sind

Permissioned Blockchains, an welchen nur zugelassene Nutzer teilnehmen dürfen. Nur diese sind berechtigt, Transaktionen auszuführen und die Daten einzusehen [26]. Dies bietet sich vor allem bei B2B-Anwendungen an, welche von verschiedenen Unternehmen genutzt werden sollen. In solchen kann es beispielsweise aufgrund von sensiblen Daten nötig sein, dass nur bestimmte Parteien Zugriff auf die Blockchain haben. Tabelle 2.1 vergleicht die Blockchaintypen untereinander sowie mit zentralen Datenbanken, da diese am häufigsten für persistente Datenspeicherung genutzt werden. Die dort erwähnten BFT-Protokolle werden genauer im Kapitel 4.2 erläutert.

	Public BC	**Permissioned BC**	**Zentrale DB**
Transaktionsdurchsatz	Gering	Hoch	Sehr hoch
Nicht vertrauenswürdige Nodes	Viele	Wenige	Keine
Konsensmechanismus	Hauptsächlich PoW	BFT-Protokolle	Keiner
Zentral verwaltet	Nein	Teilweise	Ja

Tabelle 2.1: Vergleich der Blockchaintypen mit Datenbanksystemen [2][7].

2.1.3 Exemplarische Anwendungsfälle

Die Blockchain wird als revolutionäre Technologie angepriesen (siehe [27]). Trotzdem ist es wichtig zu wissen, für welche Zwecke sie wirklich geeignet ist. Grundsätzlich ergibt eine Blockchain Sinn, wenn mehrere Parteien, welche sich nicht vertrauen, mit einem System interagieren wollen, welches von keiner dritten, zentralen Instanz verwaltet wird [2]. Um eine bessere Vorstellung von solchen Anwendungen zu erhalten, werden im Folgenden verschiedene Exemplarische Anwendungsfälle genannt und beschrieben.

Der erste Anwendungsfall, mit welchem die Blockchain-Technologie auch entstanden ist, sind Kryptowährungen. Mit Ihnen ist es möglich Geld zwischen beliebigen Parteien zu übertragen, ohne dass die Transaktionen von einer eventuell nicht vertrauenswürdigen Bank oder ähnlichem kontrolliert und verwaltet werden [10, S. x].

Weitere Anwendungsfälle ergeben sich mit der Möglichkeit, Programmlogik auf der Blockchain abzubilden. So können beispielsweise dezentrale Online-Wahlen realisiert werden. Die Stimmen würden in der Blockchain gesammelt werden und können so letztendlich nicht mehr manipuliert werden, beispielsweise von einer korrupten Regierung [28].

Ein weiterer Anwendungsfall, insbesondere für den B2B-Bereich, wäre Supply Chain Management. Über eine digitale Lieferkette sollen Material- und Informationsflüsse zu Produkten und Dienstleistungen aufgebaut und verwaltet werden [29]. Dies erlaubt Unternehmen das Automatisieren von Prozessen und das verbesserte Reagieren auf Ereignisse (z. B. Lieferverspätungen). Weiterhin ist es dem Unternehmen möglich, dem Kunden exakt aufzuzeigen, wo das Produkt und seine Unterprodukte produziert wurden. In klassischen B2B-Anwendungen müsste jedes Unternehmen, welches ein Teil der Supply Chain ist, die relevanten Daten beispielsweise durch

APIs[6] bereitstellen. Dies ist mit großen Aufwand verbunden, da diese erstmal entwickelt werden müssen. Weiterhin müssten Plattformen die Daten von unterschiedlichen Schnittstellen abfragen, um diese zusammenzuführen. Aufgrund dieses Aufwandes werden oft Dritte eingestellt, welche sich um den Aufbau und um die Datenintegration der Supply Chain kümmern. Den Aufwand sowie die eventuell nicht vertrauenswürdige dritte Partei könnte durch die Nutzung der Blockchain eliminiert werden. In dieser könnte jedes Unternehmen die relevanten Daten an einer Stelle speichern. Die Supply Chain wäre direkt in der Blockchain vorhanden und kein Unternehmen muss Datenmanipulation oder ähnliches befürchten [1].

Auch dezentrale Märkte sind für B2B-Anwendungen interessant. Der zentrale Marktplattformbetreiber (z.B. Ebay oder Amazon), welcher persönliche Informationen speichert und Gebühren für den Verkauf von Artikeln verlangt, wäre hinfällig. Nutzer könnten Waren untereinander verkaufen, während die Blockchain als Notar für den Warenaustausch dient [25].

Ein weiteres Beispiel wären Blockchain Sharing-Systeme. So könnte ein dezentrales Fahrradleihsystem aufgebaut werden. Nutzer würden mit ihrem Smartphone, über ihre in der Blockchain hinterlegte Identität (im Falle von beispielsweise Ethereum die Wallet-Adresse[7]), das Fahrrad entsperren. Dieses erkennt automatisch die gefahrene Distanz sowie die Nutzungsdauer. Über einen Smart Contract würde anschließend die automatische Zahlung erfolgen. Neben der Automatisierung besteht der Vorteil, dass mehrere Unternehmen oder auch Privatpersonen Leihfahrräder anbieten können, ohne dass sie einer zentralen Instanz mit der Verwaltung vertrauen müssen [31], [32].

2.2 Anforderungen an B2B-Anwendungen

Für den zu entwickelnden Prototypen ergeben sich verschiedene Anforderungen, welche zum Teil allgemein für B2B-(Blockchain)-Anwendungen gelten. Es ist wichtig diese zu kennen, um die Blockchain-Technologie in Bezug auf diese zu evaluieren.

Identitätsverwaltung Eine B2B-Anwendung besteht zwischen verschiedenen Unternehmen. Die Identitäten dieser müssen bekannt sein, um die Teilnahmeerlaubnis zu erteilen und Berechtigungen zu verwalten. Sie sind ebenfalls von Bedeutung, um ausgeführte Transaktionen den Teilnehmern zuordnen zu können. Die für den Prototypen genutzte Blockchain-Implementierung muss also das Registrieren und Identifizieren von Teilnehmern unterstützen.

Berechtigungsverwaltung Die verschiedenen Teilnehmer haben unterschiedliche Rechte was das Lesen, Erstellen und Verändern von Daten betrifft. So ist es beispielsweise wichtig, dass ein Teilnehmer nur seine eigenen, in der Blockchain registrierten, Assets[8] verändern kann.

[6] API: Schnittstelle für Anwendungsprogrammierung[30].

[7] Wallet: Speichert beispielsweise bei Ethereum und Bitcoin den Private Key des Nutzers und wird als Adresse für Zahlungen genutzt [8, S. 61 ff.].

[8] Assets: Besitz/Eigentum

Private Transaktionen Bei Blockchains wie Bitcoin und Ethereum sind alle Daten in der Blockchain für alle Teilnehmer einsehbar. Aufgrund von sensiblen Daten kann es allerdings vorkommen, dass nicht alle Transaktionen für alle Teilnehmer sichtbar sein sollen. So sollen beispielsweise Preisabsprachen nur von den relevanten Unternehmen eingesehen werden können.

Skalierbarkeit und Performance In Bitcoin ist lediglich ein Transaktionsdurchsatz von 7 Transaktionen pro Sekunde (TPS) möglich [7]. Hinzu kommt, dass es ca. zwischen 30 Minuten und 16 Stunden dauern kann, bis eine Transaktion bestätigt ist [33]. Darauf wird auch genauer im Kapitel 4.1 eingegangen. In B2B-Anwendungen kann die Performance und Skalierbarkeit von großer Bedeutung sein. Je nach Art der Anwendung und Anzahl der Teilnehmer muss ein höherer Transaktionsdurchsatz als bei Bitcoin erzielt werden. Die Skalierbarkeit ist vor allem beim Einsatz von IoT-Geräten von Bedeutung, welche untereinander kommunizieren und Transaktionen ausführen. Gartner behauptet, dass es bis 2020 ca. 7,5 Billionen vernetzte Geräte im Business-Sektor geben wird [34]. Damit wäre ein Transaktionsdurchsatz von 7 TPS nicht praktikabel.

Sicherheit Die Sicherheit wird durch den genutzten Konsensmechanismus realisiert. Der am häufigsten genutzte PoW ist in einem Netzwerk mit wenig Teilnehmern allerdings unsicher, da es einfach ist 51 % der Rechenleistung zu erreichen. Weiterhin führt er zu einen hohen Stromverbrauch (im Bitcoin-Netzwerk der Verbrauch von ca. 3.500.000 US-Haushalten [35]), welcher nicht erwünscht ist. Ebenfalls beeinflusst der PoW die Performance (siehe Kapitel 4).

Kapitel 3

Aktueller Stand der Technik

Die Blockchain-Technologie ist im Jahr 2008 entstanden. Jedoch befindet sie sich erst seit ca. 2016 in einem Hype, wie die Gartner Hype Cycle von 2016 und 2017 [36][37] sowie die Google Trends zum Suchinteresse [38] zeigen. Aufgrund des Hypes entstehen viele Ideen zu der möglichen Anwendung der Blockchain, darunter auch viele unnütze [2]. Im Gartner Hype Cycle ist dies die "Peak of Inflated Expectations". Es bestehen unrealistische Erwartungen an die Technologie und die Anwendungszwecke sind noch nicht klar.

Es finden sich wenig Blockchain-Anwendungen, welche bereits in Produktionsumgebungen eingesetzt werden. Dies liegt u. a. auch daran, dass es sich um eine relativ junge Technologie handelt. Viele Blockchain-Implementationen werden nur für Kryptowährungen eingesetzt. Dazu gehören beispielsweise Bitcoin, Monero, Dashcoin oder Litecoin [39]. Erst mit Ethereum wird das Konzept von Smart Contracts erfolgreich implementiert, wodurch es möglich ist eigene Programmlogik in der Ethereum-Blockchain abzubilden und so dezentrale Anwendungen zu entwickeln. So nutzt beispielsweise AXA diese für Flugversicherungen. Kommt es zu der Verspätung eines Fluges löst ein Smart Contract automatisch die Auszahlung an den Kunden aus [40]. Während es viele Ideen für dezentralisierte Anwendungen gibt, handelt es sich bei vielen nur um Prototypen.

Ein weiteres Problem an jungen Technologien ist, dass die Limitationen noch nicht komplett evaluiert und gelöst sind. Dies führt dazu, dass die Blockchain für bestimmte Anwendungen (noch) nicht geeignet ist. Die Probleme werden in diversen wissenschaftlichen Arbeiten analysiert und es werden Lösungen für diese vorgeschlagen (siehe [7], [10, S. 84] und [20]). Teilweise handelt es sich jedoch um Vorschläge, welche sich noch nicht im Einsatz befinden. Kapitel 4 beschäftigt sich genauer mit den Problemen, welche die Blockchain mit sich bringt.

Für Permissioned Blockchains haben sich verschiedene Plattformen herausgebildet. So gibt es Multichain, Quorum, Hyperledger Fabric oder Hyperledger Burrow [25]. Diese sollen die Probleme der öffentlichen Blockchains lösen. Aber auch hier gilt es die Implementationen genauer zu analysieren, um eine Aussage über ihren Nutzen zu treffen. So setzen sich Arbeiten von [25][26], [**PongnumkulPerformanceAnalysisPrivate2017**] und [41] u. a. mit der Skalierbarkeit und Performance der Permissioned Blockchains auseinander.

Dezentrale Märkte sind eine der am meisten mit Blockchain in Verbindung erwähnten Use-Cases, was an Quellen wie [25] und [42] ersichtlich wird. Neben Konzepten für diese (siehe [43]), gibt es auch Live-Systeme, wie Syscoin [44].

Dezentrale Wartungsmärkte hingegen werden nur in Verbindung mit der Supply Chain erwähnt, wie zum Beispiel in [45] oder [46]. Implementationen oder Konzeptentwürfe konnten nicht gefunden werden.

Kapitel 4

Evaluierung Permissioned Blockchains für B2B

Die Blockchain-Technologie bringt diverse Probleme mit sich, welche je nach Anwendungszweck und Blockchaintyp verschieden große Auswirkungen haben. Für den B2B-Bereich gilt es vor allem die Skalierbarkeit sowie Konsensmechanismen zu analysieren.

4.1 Skalierbarkeit

Das CAP-Theorem besagt, dass es in einem verteilten System nur möglich ist, zwei von den drei folgenden Eigenschaften zu erfüllen: Konsistenz, Verfügbarkeit und Ausfalltoleranz. Bezogen auf die Blockchain wären dies: Dezentralisierung, Skalierbarkeit und Nichtangreifbarkeit [20]. Im Zuge der Skalierbarkeit wird vor allem auf die Performance (Transaktionsdurchsatz) sowie die Anzahl von Nutzern eingegangen. Dazu erfolgt zunächst eine Analyse an aktuellen Public Blockchains und letztendlich an Permissioned Blockchains. Als Referenzwerte werden die Daten zum Transaktionsdurchsatz von IoT-Anwendungen und Plattformen genutzt. So sagt Soto., dass für Smart Cities ein Durchsatz von über 1000 TPS gebraucht wird [47]. Vandikas et al. untersuchen eine IoT-Platform und stellen eine Performance von 2173 TPS fest [**VandikasPerformanceEvaluationIoT2014a**].

4.1.1 Public Blockchains

4.1.1.1 Bitcoin

Das Bitcoin-Netzwerk erreicht aktuell einen maximalen Transaktionsdurchsatz von 7 TPS (unterschiedlich je nach Größe der Transaktionen, variiert zwischen ungefähr 0,5 und 1,2 kb [49]), bei einer Blockgröße von 1 MB. Hingegen erreichen Visa und Twitter im Durchschnitt 2000 und 5000 TPS (siehe Abb. 4.1) [10, S. 28]. Hinzu kommt, dass ungefähr 170000 unbestätigte Transaktionen[1] bestehen [50]. Berechnungen von Scherer zeigen, dass bei 11,8 Millionen Nut-

[1] Unbestätigte Transaktion: Eine Transaktion, welche noch nicht in einen Block vorkommt [8, S. 13 ff.].

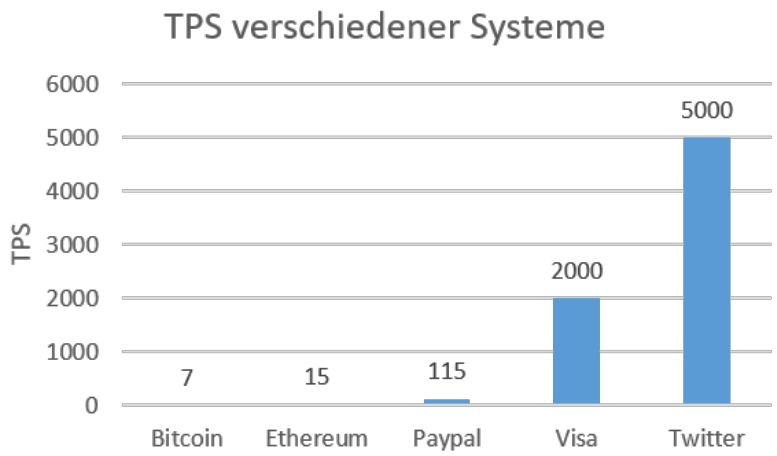

Abbildung 4.1: Vergleich der TPS bei verschiedenen Systemen [10, S. 28][48].

zern im Bitcoin-Netzwerk sowie einem Transaktionsdurchsatz von 4 TPS, jeder Nutzer nur ca. 10 Transaktionen im Jahr senden kann [20].

Der Transaktionsdurchsatz ist durch verschiedene Faktoren limitiert. Hauptsächlich durch die limitierte Blockgröße von 1 MB und dem PoW: Nur eine bestimmte Anzahl an Transaktionen passt in einen Block und nur alle 10 Minuten wird einer erstellt. Es gäbe also die Möglichkeit, die Blockgröße zu vergrößern, oder die Zeit für den PoW zu verringern, indem die Schwierigkeit angepasst wird. Es gibt jedoch diverse Nachteile, welche dadurch entstehen würden. Bei einer größeren Blockgröße würde es länger dauern, bis ein Block beim Propagieren durch das Netzwerk alle Nodes erreicht. Dies würde zu öfter vorkommenden und längeren Forks führen und somit die Sicherheit des Netzwerks beeinträchtigen. Den gleichen Effekt hätte eine kürzere PoW Zeit, da die Wahrscheinlichkeit höher ist, dass zwei Nodes zur ungefähr gleichen Zeit einen Block erstellen [20][6][51].

Entsteht ein Fork, probieren Nodes die längere und somit gültige Blockchain zu erschaffen. Gelingt dies, wird die kürzere Blockchain mit den nun sogenannten Stale Blocks verworfen. Die gesamte Rechenleistung, welche in die Stale Blocks und seine Nachfolger geflossen ist, trägt nicht zur Sicherheit des Netzwerks bei. Dies lässt sich auch anhand der Abbildung 4.2 erläutern. Innerhalb der Blockchain bestehen durch mehrere Forks 5 Branches. Das bedeutet, dass die Rechenleistung des Netzwerks auf diese aufgespalten ist. Es wird davon ausgegangen, dass 20 % der Rechenleistung in den obersten Branch geflossen ist, welcher der längste ist. Die restlichen 4 Branches erhalten je 10 % der Rechenleistung. Wenn nun ein Angreifer mit 40 % der Rechenleistung probiert, eine eigene Blockchain zu erstellen, gelingt ihm dies, da er schneller die längere Blockchain erstellen kann [51]. Zusammenfassend lässt sich sagen, dass ein Angreifer nicht 51 % der Rechenleistung für einen Angriff benötigt, wenn viele Forks bestehen [52].

Ein weiteres Problem der Forks ist, dass Miner keine Belohnung für die Arbeit an verworfenen Blöcken erhalten. Dadurch kann es zur Zentralisierung durch wachsende Mining Pools

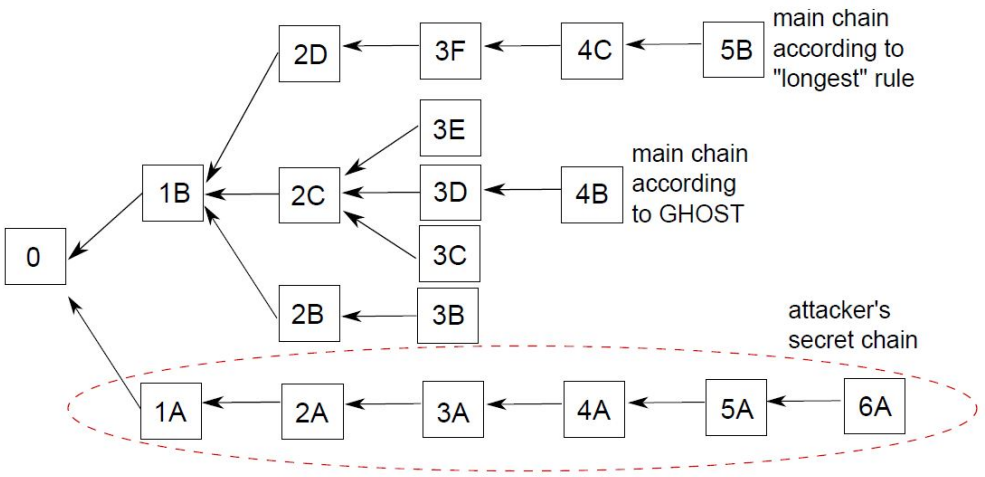

Abbildung 4.2: Auswahl der gültigen Blockchain. Bei Bitcoin die längere. Bei Ethereum die mit der meisten erbrachten Arbeit, unter Einberechnung der Uncle-Blocks [51].

kommen. Dies wird an folgenden Beispiel ersichtlich: Ein Mining Pool A besitzt 30 % der Rechenleistung, ein Mining Pool B 10 %. In dem genannten Beispiel würde Mining Pool A in 70 % aller Fälle einen Stale Block erzeugen und B in 90 % aller Fälle. Kein Miner würde dem Mining Pool B beitreten, da die Wahrscheinlichkeit geringer ist, dass B gültige Blöcke erschafft. A hingegen würde immer mehr Miner und somit mehr Rechenleistung erhalten [6].

An dieser Stelle sollte auch darauf hingewiesen werden, dass schnellere Blockerstellungszeiten nicht zwingend zu schnelleren Transaktionsbetätigungen führen. Transaktionen werden zwar schneller in Blöcken aufgenommen, aber es muss auf mehr Nachfolger gewartet werden, um sicher zu gehen, dass die Transaktion nicht in einem Fork vorkommt [20].

4.1.2 Ethereum

Bessere Performance durch GHOST Das Ethereum Netzwerk nutzt das sogenannte GHOST-Protokoll und erreicht damit eine Transaktionsdurchsatz von 15 TPS, bei einer durchschnittlichen Zeit von 15 Sekunden um den PoW zu erbringen. Dieses löst Probleme des Forkings und der Benachteiligung von Minern. Ersteres wird dadurch gelöst, dass Stale Blocks in die Berechnung der gültigen Blockchain einfließen. Anders als bei Bitcoin, wo lediglich die Parents und deren Nachfolger eine Rolle spielen. Die Stale Blocks werden in Ethereum als “Uncles” bezeichnet. Kurz gesagt, ist ein Uncle ein alternativer gefundener Block, welcher auf der gleichen Höhe wie der Parent bestehen würde [6].

Die Bestimmung der gültigen Blockchain wird an der Abbildung 4.2 ersichtlich. In Ethereum ist die Blockchain die gültige, für welche die meiste Arbeit aufgebracht wurde, unter Einbezug der Uncles. Das führt dazu, dass der Branch mit den meisten Uncles bestehen bleibt. Das bedeutet letztendlich, dass die gesamte Rechenleistung das Netzwerk absichert, auch wenn diese sich auf die Branches aufteilt. Ein Angreifer braucht somit weiterhin 51 % der Rechenleistung um einen Angriff auszuführen [51].

Damit ist allerdings noch nicht das Problem der Zentralisierung durch Mining Pools gelöst. Es besteht weiterhin keine Motivation für Miner, Uncles zu minen. Deswegen ist das GHOST-Protokoll in Ethereum so erweitert, dass Miner Ether[2] als Belohnung für das Erstellen von Uncles erhalten (allerdings weniger als bei vollwertigen Blöcken). Somit besteht ebenfalls die Motivation, kleineren Mining Pools beizutreten [6]. An dieser Stelle sollte auch erwähnt werden, dass Miner entscheiden können, an welchem Branch sie arbeiten [7].

Während Ethereum die Probleme löst, welche durch Forks entstehen, ist der Transaktionsdurchsatz trotzdem limitiert. Die Blockgröße muss klein genug bleiben, damit das Propagieren im Netzwerk effizient bleibt [20]. Ansonsten würden Miner unter Umständen so benachteiligt werden, dass sie nur sehr selten bis gar nicht den aktuellen Block der gültigen Blockchain erhalten würden. Dies wiederum würde dazu führen, dass sie nur Uncles minen können und so nie die volle Belohnung erhalten können. Hinzu kommt, dass Uncles nur gültig sind, wenn sie maximal eine bestimmte Anzahl an Generationen vom aktuellen Block in der gültigen Blockchain entfernt sind. Ansonsten hätten die Miner auch weniger Motivation ehrlich zu bleiben, da sie ohne Nachteile an der Blockchain eines Angreifers arbeiten könnten [6].

Schlechtere Performance durch Smart Contracts Ethereum löst die Probleme von häufig auftretenden Forks und erlaubt so einen höheren Transaktionsdurchsatz sowie schnellere Transaktionsbestätigungszeiten. Weitere Probleme entstehen jedoch, wenn eine Blockchain nicht nur Geldtransfertransaktionen verarbeitet. Ethereum erlaubt das Speichern und ausführen von eigenem Code durch Smart Contracts. Dadurch steigt die Komplexität der auszuführenden Transaktionen. Die Performance nimmt ab, da die entstehenden größeren Blöcke eine längere Propagationszeit verursachen. Ebenfalls verschlechtert sich die Performance des Netzwerks, da die Daten schwieriger zu verarbeiten sind. So muss jede Node alle Transaktionen verifizieren, Smart Contract-Code ausführen und die Ergebnisse speichern [20].

In Ethereum werden Transaktionen sequentiell bei allen Nodes ausgeführt. Dazu gehört das Ausführen von Smart Contract-Code sowie das Verifizieren der Ergebnisse. Nur so können in Konflikt stehende Transaktionen, wie zum Beispiel beim Double-Spend, erkannt werden. Eine Parallelausführung ist nicht möglich. Dies verschlechtert letztendlich die Performance des Netzwerks, da es länger dauert Transaktionen auszuführen [20]. Dies wird auch durch ein Beispiel klar. Ein Angreifer kann DoS-Angriffe ausführen, indem er komplex auszuführende Smart Contracts schreiben. Die Ausführung von diesem bei jeder Node führt dazu, dass keine anderen Operationen ausgeführt werden können. Ethereum löst dieses Problem, indem der Transaktionssender für jeden Berechnungsschritt eine Gebühr zahlen muss. Dies funktioniert jedoch nur, wenn in der Blockchain-Anwendung eine Kryptowährung genutzt wird [41].

Ebenfalls behauptet Vukolic, dass der Code der Smart Contracts nicht bei allen Nodes ausgeführt werden muss. Um Konsens zu erreichen, genügt es, dass alle Nodes den gleichen Stand der Daten erhalten. Deshalb könnte die Codeausführung nur von bestimmten Nodes ausgeführt

[2]Ether: Kryptowährung von Ethereum [6].

werden. Das Problem dabei ist, dass eine genügend große Anzahl an vertrauenswürdigen Teilnehmer festgelegt werden muss [41]. Damit geht allerdings auch das vertrauenslose Modell der Blockchain verloren.

Letztendlich lässt sich sagen, dass Public Blockchains nicht skalieren . Um dies zu lösen, müssen schnellere Blockpropagationszeiten, geringere Transaktionsgrößen und/oder bessere Transaktionsverarbeitung realisiert werden [20]. Bei Betrachtung des CAP-Theorem wird ersichtlich, dass nur die Eigenschaften Dezentralisierung und Sicherheit gegeben sind. Es ist jedoch zu bedenken, dass viele Probleme der Skalierbarkeit aufgrund des genutzten Konsensmechanismus bestehen. Auch wenn es teilweise Lösungsvorschläge für diese gibt, genügen sie bisher nicht um Skalierbarkeit herzustellen. Deshalb gilt es, die Limitationen von Permissioned Blockchains sowie alternative Konsensmechanismen für diese zu analysieren.

4.1.3 Permissioned Blockchains

Permissioned Blockchains werden eingesetzt, wenn nur bestimmte Teilnehmer an der Blockchain teilnehmen sollen. So gibt es beispielsweise auch eine verwaltende Instanz, welche u. a. die ersten berechtigten Teilnehmer bestimmt und die Netzwerkkonfiguration vornimmt. Dadurch entsteht eine stärkere Zentralisierung als bei Public Blockchains. Bezogen auf das CAP-Theorem, müssten sich dadurch die Sicherheit und/oder Skalierbarkeit verbessern. Dies führt allerdings auch dazu, dass ein größeres Maß an Vertrauen zwischen den Teilnehmern gegeben sein muss. Dies wird zum Teil dadurch sichergestellt, dass jeder Teilnehmer die Rechte zur Teilnahme am Netzwerk erhalten hat und die Identitäten dieser bekannt sind. Durch letztere ist nachverfolgbar, welche Teilnehmer welche Transaktionen ausführen [20].

Scherer behauptet, dass das größere Vertrauen es erlaubt, den Nodes verschiedene Aufgaben zuzuteilen. Dies beschreibt er am Beispiel von Hyperledger Fabric (im folgenden nur noch Fabric genannt), einer Permissioned-Blockchain. In dieser gibt es Peer und Ordering Nodes. Erstere simulieren das Ausführen von der Transaktionen und der damit verbundenen Datenänderungen. Letztere bestimmen die Reihenfolge der auszuführenden Transaktionen in den Blöcken. Peer Nodes führen die bereits simulierten Transaktionen nacheinander aus und erkennen Konflikte in den Transaktionen (genauer im Kapitel 5.3 erklärt). Die Ordering Nodes sind also letztendlich für den Konsens verantwortlich. In Gegensatz zu Ethereum können Peer Nodes so parallel Transaktionen verarbeiten. Sie müssen sich nicht um eventuelle Konflikte oder die Reihenfolge der Transaktionen kümmern. Letztendlich würde die Performance (im Bezug auf die Transaktionsverarbeitung), von der Hardware der Peers und der Anzahl dieser abhängen.

Scherer führt ebenfalls Tests durch, um die Performance von Fabric zu analysieren. Dazu nutzt er eine frühe und unstabile Version 1.0. Das Netzwerk besteht aus einer Ordering und einer Peer Node. Es wird kein Konsensmechanismus genutzt. Die Anwendung selbst unterstützt die Zahlung mittels digitaler Assets (z. B. Tokens bzw. Coins) zwischen zwei Accounts. Dabei erreicht er einen maximalen Transaktionsdurchsatz von 350 TPS. Dabei ist allerdings zu bedenken, dass der Test auf einer Maschine mit limitierten Ressourcen ausgeführt wird. Um einen

maximalen Transaktionsdurchsatz zu erreichen, müssten mehrere leistungsstarke Computer für den Test eingesetzt werden. Scherer stellt ebenfalls fest, dass der Transaktionsdurchsatz abnimmt, desto mehr Peers es gibt, welche Transaktionen bestätigen. Dies liegt daran, dass diese sogenannten Endorser untereinander kommunizieren müssen. Pro Node müssten $O(n^2)$ Nachrichten gesendet werden, wobei n die Anzahl an Nodes ist. Die Anzahl an effizient nutzbaren Endorsern ist also beschränkt. So führt der Test mit einem Endorser zu einen Transaktionsdurchsatz von ca. 70 TPS, während mit 14 Endorsern nur noch ca. 15 TPS erreicht werden. Somit wäre die Anzahl an effizient nutzbaren Endorsern begrenzt. Scherer behauptet jedoch, dass Fabric mit rechenstarken Maschinen in der Theorie tausende TPS unterstützt [20].

Ein Paper von Pongnumkul et al. vergleicht die Leistung von Fabric mit Ethereum. Er nutzt dazu die stabile Version 0.6. Er führt die Tests ebenfalls mit nur einer Peer Node durch. Zur Ordering Node macht er keine Angabe. Die Anwendung ist die gleiche wie bei Scherer und es wird ebenfalls kein Konsensmechanismus genutzt. Pongnumkul stellt fest, dass die Performance von Fabric in allen Kriterien besser ist als bei Ethereum. So betrug die Zeit, bis eine Beispieltransaktion verarbeitet wurde 41 Sekunden. Ethereum hingegen benötigte 478 Sekunden. Tests zum maximalen Transaktionsdurchsatz haben ergeben, dass Ethereum 40 TPS und Fabric 300 TPS erreicht hat. Die dazugehörige Abbildung 4.3 zeigt auch, dass die Unterschiede zwischen den beiden Systemen signifikanter sind, desto mehr Transaktionen verarbeitet werden müssen [**PongnumkulPerformanceAnalysisPrivate2017**].

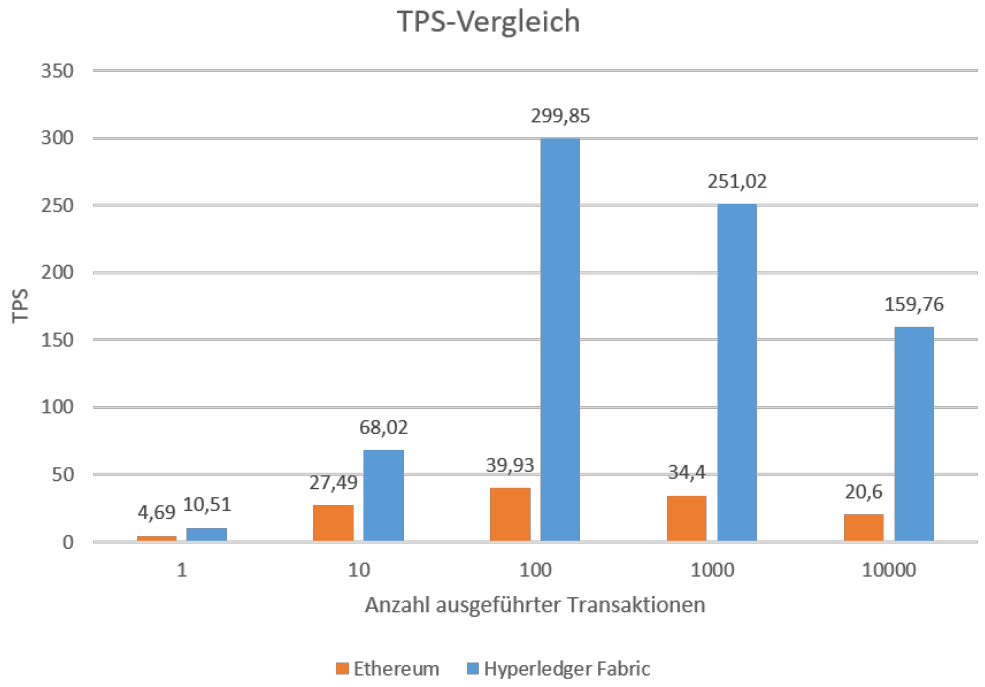

Abbildung 4.3: Vergleich des Transaktionsdurchsatzes von Ethereum und Fabric [**PongnumkulPerformanceAnalysisPrivate2017**].

Bei beiden Tests ist zu bedenken, dass sie mit nicht aktuellen Versionen von Fabric ausgeführt wurden. Mittlerweile gibt es eine stabile Version 1.0 sowie eine Preview von Version 1.1.0

[53]. Es ist also möglich, dass die Performance sich mittlerweile verbessert hat.

Letztendlich lässt sich sagen, dass Permissioned Blockchains, was die Verarbeitung von Transaktionen betrifft, eine bessere Performance erzielen. Diese wurde allerdings noch nicht unter der Nutzung verschiedener Konsensmechanismen betrachtet. Weiterhin muss das CAP-Theorem noch auf die Sicherheit untersucht werden. Deshalb erfolgt im nächsten Kapitel die Analyse der Skalierbarkeit und Sicherheit von Konsensmechanismen.

4.2 Konsensmechanismen

Das Erreichen von Konsens in einer Blockchain, ist eine Abwandlung des Byzantine Generals Problem. In diesem gibt es Generäle, welche Armeen kommandieren, welche eine Stadt umzingeln. Ein Angriff auf diese ist nur erfolgreich, wenn alle Armeen gleichzeitig angreifen. Die Generäle müssen untereinander kommunizieren und Konsens darüber herstellen, ob ein Angriff erfolgen soll. Allerdings gibt es Verräter unter diesen. Es handelt sich also um ein vertrauensloses Umfeld. Genau so kann es in einer Blockchain zu den sogenannten Byzantine Faults kommen, wenn nicht vertrauenswürdige Nodes Daten manipulieren können. Deshalb müssen verteilte Systeme Byzantine Fault Tolerance (BFT) herstellen, um eine gewisse Anzahl an nicht vertrauenswürdigen Teilnehmern zu tolerieren. Dies geschieht in Blockchains über die Konsensmechanismen [7].

Eine Blockchain, welche den PoW als Konsensmechanismus nutzt, ist nicht skalierbar. Ebenfalls würde er in Netzwerken mit relativ wenig Teilnehmern die Sicherheit beeinträchtigen, da ein Teilnehmer einfacher 51 % der Rechenleistung erreichen kann. Für Permissioned Blockchains muss also ein Konsensmechanismus gefunden werden, welcher Skalierbarkeit und Sicherheit herstellt. Scherer behauptet, dass aufgrund des höheren Vertrauens in Permissioned Blockchains, ein Konsensmechanismus genutzt werden kann, welcher Vertrauen in geringerem Maße als der PoW garantiert. Somit könnten Skalierbarkeit und Sicherheit hergestellt werden [20]. Im Folgenden werden verschiedene Konsensmechanismen verschiedener Blockchain-Technologien miteinander verglichen.

4.2.1 Proof of Stake

Beim PoS hängt die Wahrscheinlichkeit, einen Block zu minen, von der Menge des Eigentums (z. B. Kryptowährung) eines Nutzers ab. Desto mehr Bitcoins dieser beispielsweise besitzt, desto höher ist die Wahrscheinlichkeit, für das Mining ausgewählt zu werden. Ähnlich wie beim PoW könnte ein Teilnehmer mit 51 % aller Bitcoins das Netzwerk angreifen, da er die längere Blockchain erstellen kann. Aber selbst wenn es ein Teilnehmer schafft, 51 % der Bitcoins zu besitzen, hätte er keine Motivation dazu. Denn letztendlich würde ein Angriff den Kurs von Bitcoin senken und somit würde der Miner sich selbst schaden [7]. Da der PoS hauptsächlich nur bei Kryptowährungen genutzt werden kann, ist er für Permissioned Blockchains uninteressant.

4.2.2 Proof of Elapsed Time

Der PoET wird in Intels Blockchain-Technologie Hyperledger Sawtooth genutzt. Die grundlegende Idee ist, dass eine Node eine zufällige Zeit generiert, welche Sie warten muss, um einen Block zu erstellen. Um sicherzustellen, dass die generierte Zeit nicht verfälscht wurde, wird Trusted Computing [3] genutzt. So stellen Intels Software Guard Extensions (SGX) sicher, dass Code nicht modifiziert werden kann. Eine Node muss also über solchen unmodifizierbaren Code eine Zeit generieren. Weiterhin erfolgen statistische Tests, um zu verhindern, dass eine Node Blöcke zu schnell und somit zu oft erstellt. Letztendlich ist die Blockerstellung damit fair verteilt und kein Teilnehmer kann die Blockchain kontrollieren. Im Prinzip funktioniert der Mechanismus wie der PoW. Dort wird eine Wartezeit durch das Finden eines Hashes sichergestellt, während dies beim PoET durch die Hardware sichergestellt wird. Dadurch, dass es keine rechenintensiven Aufgaben gibt, ist die Performance sichergestellt. Es ist jedoch zu bedenken, dass es bisher wenige Analysen zu der Sicherheit des PoET gibt. Ein Paper von Chen et al. stellt fest, dass der Konsensmechanismus unter bestimmten Umständen unsicher ist, schlägt aber auch Lösungen dafür vor. Ebenfalls kommt hinzu, dass der Hardware von Intel für das Trusted Computing vertraut werden muss [55].

4.2.3 Diversity Mining Consensus

Der Diversity Mining Consensus ist ähnlich dem PoW. In diesem ist der Block einer Node nur valide, wenn sie eine bestimmte Anzahl an vorherigen Blöcken nicht erstellt hat. Dies führt letztendlich dazu, dass eine Wartezeit für jede Node nicht durch das Finden eines Hashes (siehe PoW) oder durch eine generierte Zeit (siehe PoET) realisiert wird. Für den Mechanismus müssen alle Teilnehmer am Netzwerk bekannt sein. Auch hier kann es zu Forks kommen, wenn zwei Nodes zur ungefähr gleichen Zeit einen Block erstellen. Ebenfalls wird die entstehende längere Blockchain akzeptiert. Durch die Wartezeit wird sichergestellt, dass der Branch mit den meisten Minern die längere Blockchain erstellt. Die Gefahr bei diesem Konsensmechanismus ist, dass bösartige Miner sich zusammen tun können. Unter den richtigen Umständen erstellt der Zusammenschluss die meisten Blöcke und kann somit auch bei Forks eine längere Blockchain erstellen, um so Double-Spending-Angriffe auszuführen. Die Performance ist aufgrund des fehlenden Rechenaufwands sichergestellt [56][57].

4.2.4 QuorumChain

Quorum ist ein Fork von Ethereum für Permissioned Blockchains. Dieser nutzt den Konsensmechanismus QuorumChain. Es gibt Voter und Block-Maker Nodes. Die Block-Maker schlagen Blöcke zum Erstellen vor und die Voter stimmen für diese ab. Um Forks zu verhindern, warten die Block-Maker eine zufällige Zeit und erstellen den Block. Die Voter validieren diesen und

[3] Trusted Computing: Hardware, welche sicherstellt, dass Software sich wie erwartet verhält [54].

stimmen für ihn ab, indem Sie u. a. die Transaktionen ausführen und überprüfen, ob der vorherige Block genug Stimmen erhalten hat. Die Nodes hängen letztendlich den Block an ihre lokale Blockchain an, welcher einen bestimmten Schwellwert an Stimmen überschritten hat oder im Falle eines Forks die meisten Stimmen erhalten hat. Jedoch ist die Sicherheit des Mechanismus fragwürdig. Laut Cachin et al. soll es schon allein mit einem bösartigen Block-Maker zu Forks kommen, welche Double-Spending-Angriffe ermöglichen und die Zeit erhöhen, bis eine Transaktion nicht mehr verworfen werden kann [57].

4.2.5 Practical Byzantine Fault Tolerance

Der PBFT gehört zu der Familie der BFT-Protokolle. Hier müssen ebenfalls alle Teilnehmer bekannt sein. Beim PBFT wählen die Teilnehmer eine Leader-Node. Jede Runde schlägt diese einen neuen Block mit auszuführenden Transaktionen vor. Dieser wird an alle anderen Nodes weitergeleitet. Anschließend wird der Konsens hergestellt. $^2/_3$ der Nodes müssen dem im Block enthaltenen Transaktionen zustimmen, damit er erstellt wird. Erst dann werden die Transaktionen bei jeden Teilnehmer ausgeführt. Deshalb können bis zu $^1/_3$ der Nodes unvertrauenswürdig sein. Ein Angreifer müsste für einen Angriff die Kontrolle über $^2/_3$ der Nodes haben [23][7].

Vukolic behauptet, dass es BFT-Protokolle gibt, welche einen Transaktionsdurchsatz von mehreren 10000 TPS unterstützen. Die Skalierbarkeit dieser bezüglich der Anzahl an Nodes ist jedoch begrenzt [58]. Croman erzielt bei seinen Tests mit dem PBFT, bei 8 Nodes und 8192 auszuführenden Transaktionen einen Transaktionsdurchsatz von 14000 TPS. Weiterhin wird ersichtlich, wie die Performance mit der Anzahl an Nodes abnimmt. Mit 64 Nodes und 8192 auszuführenden Transaktionen wird ein Transaktionsdurchsatz von 4500 TPS erreicht [59]. Im Gegensatz zum PoW besteht hier eine bessere Skalierbarkeit bezüglich des Transaktionsdurchsatzes, allerding ist sie bezüglich der Anzahl an Teilnehmern begrenzt [58].

Ein weiterer Vorteil von BFT-Protokollen ist, dass es Consensus Finality gibt. Das bedeutet, dass es nicht zu Forks kommen kann. Somit müssten Nutzer nicht darauf warten, dass mehrere Blöcke nach einer Transaktion erstellt werden, damit die Sicherheit gegeben ist, dass diese endgültig bestehen wird. Somit entfällt auch die Gefahr von Double-Spending-Angriffen [58].

Die Angriffe, welche mit mehr als $^1/_3$ der Voting Power erfolgen können, sind die sogenannten Censorship Attacks. So könnten Nodes verhindern, dass neue Blöcke entstehen, indem sie ihre Stimme zurückhalten. Weiterhin könnten sie bestimmte Transaktionen zensieren, indem sie dafür stimmen, diese nicht in einem Block aufzunehmen [60].

4.2.6 Tendermint

Tendermint ist eine Abwandlung des PBFT-Konsensmechanismus. Der größte Unterschied besteht darin, dass die Nodes, welche einen Block vorschlagen, in einem Round-Robin Verfahren ausgewählt werden. So gibt es in jeder Runde einen neuen Blockersteller. Hinzu kommt, dass die Teilnehmer mit ihren Coins, welche die Voting Power bestimmen, abstimmen. Die Coins werden

für eine bestimmte Zeit an eine Vote gebunden und damit für die weitere Nutzung gesperrt. Aufgrund der Asynchronität des Netzwerks kann es vorkommen, dass eine Node noch keine Information über einen bereits neu erstellten Block erhalten hat. Das führt dazu, dass er einen neuen Block auf der gleichen Höhe des bestehenden Blocks vorschlägt. Es kann dann zu einen Fork kommen, wenn beide Blöcke $^2/_3$ der Voting Power erhalten. Dies ist jedoch nur möglich, wenn mindestens $^1/_3$ der Voting Power bösartig genutzt wird, also doppelt abstimmt. Durch die an die Abstimmung gebundenen Coins, können diese jedoch bestraft werden. So werden die gebundenen Coins einfach zerstört [61][62]. Es ist zu bedenken, dass der Mechanismus nur mit einer Kryptowährung im vollen Umfang funktioniert.

4.2.7 Sonstige BFT-Konsensmechanismen

Neben den bisher erwähnten BFT-Konsensmechanismen gibt es noch diverse andere, welche das Prinzip des PBFT mehr oder weniger abwandeln. So gibt es Symbiont und R3 Corda mit BFT-SMaRt, Iroha mit Sumeragi, Kadena mit ScalableBFT und Chain mit Federated Consensus [57]. Aufgrund der Vielfalt und Detailreichheit jeder dieser Konsensmechanismen, wird nicht genauer auf sie eingegangen. Es genügt die Prinzipien von BFT-Protokollen anhand des PBFT zu verstehen.

4.3 Sonstige Einschränkungen

4.3.1 Private Transaktionen

In Blockchains wie bei Bitcoin und Ethereum ist es nicht möglich, private Transaktionen auszuführen. Das bedeutet, alle Transaktionen sind für alle Teilnehmer sichtbar. In Permissioned Blockchains kann es Fälle geben, wo dies nicht erwünscht ist. So soll beispielsweise eine Preisabsprache zwischen 2 Teilnehmern in der Blockchain dokumentiert werden, welche für andere nicht ersichtlich ist.

In Quorum wird dies realisiert, indem in einer Transaktion angegeben wird, welche Teilnehmer die Transaktion sehen dürfen. Diese wird anschließend verschlüsselt und kann nur von den angegebenen Teilnehmern entschlüsselt werden. Die Transaktion wird im Netzwerk verteilt und nur von den Teilhabern ausgeführt [63].

In Fabric werden private Transaktionen über Channels ermöglicht. Dabei ist jeder Channel eine eigene Blockchain, mit verschiedenen Teilnehmern. So würde es beispielsweise einen öffentlichen Channel geben, an welchen alle Teilnehmer der Permissioned Blockchain teilnehmen. Zusätzlich würde es private Channel geben, welche nur zwischen bestimmten Teilnehmer bestehen würden. Keine Daten können zwischen den Channeln übertragen werden [20].

Der Konsens bei privaten Transaktionen stellt ein Problem dar. In Quorum sind Teile der Transaktion verschlüsselt, wodurch diese nicht von allen Nodes verifiziert werden können. Die Verifizierung ist jedoch trotzdem anhand anderer Daten möglich (siehe [64]). In Fabric wird

der Konsens über Ordering Nodes hergestellt. Teilnehmer eines Channels könnten bereits bestehende Ordering Nodes nutzen, wodurch diese allerdings die privaten Transaktionen erhalten. Ebenfalls könnten für den Channel neue Ordering Nodes von jedem Teilnehmer erstellt werden. Mit 2 Teilnehmern wäre dies jedoch je nach genutzten Konsensmechanismus problematisch. Zu dem Konsens bei privaten Transaktionen in Fabric konnten keine Quellen gefunden werden.

4.3.2 Datenmenge

Ein noch nicht angesprochenes Problem der Blockchain ist die Datenredundanz im Bezug auf die Datenmenge. Da keine Daten in der Blockchain gelöscht werden können, wächst sie stetig an. So ist die Bitcoin-Blockchain im Moment 151 GB groß [65]. Größere Datenmengen werden schneller erreicht, wenn nicht nur Geldtransferaktionen bestehen.

Die Datenmenge stellt ein Problem dar, da Teilnehmer ab einen bestimmten Punkt eventuell nicht mehr bereit sind die Blockchain auf ihrer Node zu speichern. Dies würde zu weniger Minern und somit zu höherer Zentralisierung führen [20].

4.4 Zusammenfassung

Die Skalierbarkeit von Public Blockchains stellt ein Problem für Anwendungen dar. Die Performance ändert sich mit wachsender Nutzerzahl zwar nicht [20], ist aber trotzdem zu gering, um eine Vielzahl an Nutzern zu unterstützen. In Permissioned Blockchains ist das Gegenteil der Fall. Es besteht eine höhere Performance, welche jedoch mit der Anzahl an Peers abnimmt. Letztendlich ist unklar, in welchem Maße Skalierbarkeit mit Permissioned Blockchains erreicht werden kann. Die Transaktionsverarbeitung von Fabric skaliert mit der genutzten Hardware. Allerdings verringert sich die Performance, desto mehr Nodes untereinander kommunizieren müssen. Konsensmechanismen wie PBFT unterstützten tausende von TPS pro Sekunde. Der Transaktionsdurchsatz nimmt jedoch ebenfalls mit der Anzahl an Nodes ab. Zukünftige Analyse ist nötig, um festzustellen, in welchen Maße die Performance beeinflusst wird. Letztendlich kann so keine eindeutige Aussage zu der Skalierbarkeit und zum CAP-Theorem getroffen werden. Es kann davon ausgegangen werden, dass Anwendungen basierend auf Permissioned Blockchains auf einen bestimmten Transaktionsdurchsatz bzw. auf eine bestimmte Anzahl an Nodes beschränkt sind. Die von Soto und Vandikas genannten tausenden TPS für IoT-Anwendungen können mit Fabric in der Theorie erreicht werden.

Blockchain-Implementationen wie Fabric und Quorum zeigen, dass private Tramsaktionen realisiert werden können. Bezüglich dieser ist jedoch unklar, wie effizient der Konsens umgesetzt werden kann. Für die redundante Datenhaltung und damit zusammenhängende Datenmenge, welche jeder Miner speichern muss, gibt es keine Lösung.

Kapitel 5

Umsetzung eines dezentralen Wartungsmarktes

Basierend auf den Erkenntnissen aus dem letzten Kapitel, wird im Folgenden das Konzept des dezentralen Wartungsmarktes vorgestellt und umgesetzt.

5.1 Konzept und Anforderungen

Als Proof-of-Concept erfolgt die Entwicklung einer prototypischen B2B-Blockchain-Applikation in Form eines automatisierten sowie dezentralisierten Wartungsmarktes. Teilnehmer an diesem sind multiple Unternehmen und Wartungsanbieter. Erstere besitzen IoT-Geräte, welche erkennen können, dass sie eine Wartung benötigen. Die Wartungsanbieter erhalten die Informationen zur Wartung und können sich für diese anmelden. Anschließend würden sie diese durchführen und dabei die Wartungsschritte loggen.

In klassischen B2B-Anwendungen wäre die Realisierung dieses Systems auf zwei Arten erfolgt. Bei Ersterer gäbe es eine dritte Partei, welche den Markt verwaltet und bei welcher sich alle Unternehmen und Wartungsanbieter anmelden müssen (z. B. Ebay). Die andere Möglichkeit wäre, dass eines der teilnehmenden Unternehmen den Markt verwaltet. Bei beiden Optionen müssten die Teilnehmer ihre Daten einer eventuell nicht vertrauenswürdigen zentralen Instanz zur Verfügung stellen.

Um dies zu verhindern, wird der Wartungsmarkt auf Basis der Blockchain-Technologie implementiert. Somit können beliebig viele Unternehmen und Wartungsanbieter an dem System teilnehmen, ohne dass eine Datenmanipulation durch die Teilnehmer oder eine zentrale Instanz befürchtet werden muss. Wartungen werden verfolgbar und unveränderbar dokumentiert, ohne dass Vertrauen zwischen den Parteien nötig ist.

Neben den im Kapitel 2.2 genannten allgemeinen Anforderungen an B2B-Blockchain-Anwendungen, ergeben sich speziellere für das zu entwickelnde System. Die Spezifizierung dieser ist wichtig, denn auf Basis dieser wird eine Blockchain-Plattform ausgewählt und auf verschiedene Probleme analysiert. Folgende Anforderungen ergeben sich für den dezentralen

Wartungsmarkt:

Automatische Wartungsankündigung Wenn die IoT-Geräte Wartungsbedarf erkennen, kündigen sie dies in Form eines Smart Contracts in der Blockchain an. Es kann verschiedene Gründe für die Wartung geben. So kann beispielsweise ein Wartungsdatum erreicht werden oder Sensorwerte weisen auf einen Fehler hin.

Konditionen für Vertragsannahme Wartungsanbieter können die Smart Contracts nur unter bestimmten Konditionen annehmen. So muss beispielsweise ein Wartungsanbieter bereits Erfahrungen mit der Wartung von bestimmten Geräten haben. Diese Information könnte ebenfalls aus der Blockchain abgefragt werden. Weiterhin darf der Vertrag beispielsweise noch nicht von einem anderen Anbieter akzeptiert worden sein.

Wartungs-Logging Die Wartungsanbieter melden sich am Gerät mit an und loggen die durchgeführten Wartungsschritte. So entsteht eine nachverfolgbare und nicht löschbare Historie an durchgeführten Wartungen mit dazugehörigen Schritten.

Wartungsüberprüfung Die IoT-Geräte überprüfen, ob eine Wartung korrekt erfolgt ist und schließen den dazugehörigen Wartungsvertrag. Die Überprüfung kann anhand der geloggten Schritte sowie Sensorwerten erfolgen, welche vor und nach der Wartung existiert haben.

5.2 Technologieauswahl

Aus den Anforderungen an den dezentralen Wartungsmarkt ergeben sich die folgenden Anforderungen an die zu nutzende Plattform:

- Möglichkeit Permissioned Blockchains zu erstellen
- Möglichkeit eigene Programmlogik zu implementieren (Smart Contracts)
- Höchstmögliche Performance (Transaktionsdurchsatz)
- Höchstmögliche Skalierbarkeit im Bezug auf die Anzahl der Teilnehmer
- Konsensmechanismus mit höchstmöglicher Sicherheit und Performance
- Private Transaktionen
- Mindestens Version 1
- Gute Dokumentation und Community Support

Zunächst einmal sind öffentliche Blockchain-Plattformen, wie Bitcoin, Ethereum und Sawtooth Lake aus der Auswahl entfallen. Daraufhin wurden die Permissioned Blockchains, aufgelistet in der Tabelle 5.1, miteinander verglichen. Multichain, OpenChain sowie Chain Core konnten ausgeschlossen werden, da sie keine Smart Contracts unterstützen. Die Plattformen mit den höchsten Transaktionsdurchsatz sind Hyperledger Fabric und Hyperledger Burrow. Burrow

befindet sich jedoch noch in einer frühen Version, wodurch es ebenfalls nicht zur Auswahl steht [53]. Letztendlich steht so nur noch Fabric zur Auswahl.

Version 1 ist bereits im Juli 2017 erschienen [53]. Fabric bietet eine umfassende Dokumentation sowie Community Support über RocketChat und StackOverflow [66]. Private Transaktionen werden über Channels realisiert [20]. Ein großer Vorteil von Fabric gegenüber anderen Plattformen, sind austauschbare Konsensmechanismen. Dadurch, dass es keinen festgelegten Konsensmechanismus gibt, kann je nach Use-Case ein Konsensmechanismus ausgewählt werden, welcher die benötigte Performance, Skalierbarkeit und Sicherheit herstellt [41]. Dies ist vor allem wichtig im Prototyping. Wenn der Prototyp vom entstehenden dezentralen Wartungsmarkt erweitert werden soll (z. B. um mehr Teilnehmer), kann ein neuer Konsensmechanismus gewählt werden, welcher den neuen Anforderungen entspricht. Vukolic nennt ebenfalls den Vorteil, dass Fabric eine bessere Performance als andere Plattformen erzielt, da die Nodes nach Peer und Ordering Nodes aufgeteilt werden. Aufgrund dessen behauptet er auch, dass Fabric die Limitationen anderer Permissioned Blockchains löst [41]. Somit wird letztendlich Fabric für die Umsetzung des dezentralen Wartungsmarktes genutzt.

Unternehmen	**Technologie**	**Performance**	**Smart Contracts**
Coin Sciences	Multichain	100-1000 TPS	Nein
J.P. Morgan	Quorum	12-100 TPS	Ja
IBM	Hyperledger Fabric	10k-100k TPS	Ja
Coinprism	OpenChain	1000+ TPS	Nein
Chain	Chain Core	N/A	Nein
R3	Corda	N/A	Ja
Monax	Hyperledger Burrow	10k TPS	Ja

Tabelle 5.1: Vergleich diverser Permissioned Blockchain Plattformen [25][67]

5.3 Hyperledger Fabric und Composer - Grundlagen

5.3.1 Hyperledger Fabric

Hyperledger Fabric ist eine Blockchain-Plattform für Business-Netzwerke. Es ist darauf ausgelegt modular (z. B. austauschbare Konsensmechanismen) zu sein, um es einfach erweitern und somit für möglichst viele Use-Cases nutzbar machen zu können [9]. Im Folgenden wird das grundlegende Konzept von Fabric erklärt.

Chaincode Fabric erlaubt den Teilnehmern das Erstellen, Interagieren und Nachverfolgen von digitalen Assets. Diese bestehen letztendlich aus Ansammlungen von Key-Value-Paaren. Für die Interaktion werden Transaktionen genutzt. Die Assets und Transaktionen sind u. a. im

Chaincode definiert. Dieser ist letztendlich bei den Nodes im Netzwerk installiert [20]. Da der Chaincode Programmlogik abbildet, kann er auch als Smart Contract bezeichnet werden [68].

Identitätsverwaltung Jede Node im Netzwerk muss eine Identität erhalten. Nur so können die Teilnehmer die Daten lesen und Transaktionen ausführen [20]. Die Registrierung sowie das Erstellen von Zertifikaten wird von einer Certificate Authority (CA) übernommen. Die Teilnehmer selbst können CAs sein. So würde zum Beispiel jedes Unternehmen Identitäten und Zertifikate für seine Mitarbeiter erstellen [69].

State Database Jede Node speichert die Blockchain und zusätzlich eine sogenannte State Database. Diese speichert den aktuellsten Status der digitalen Assets. Anders formuliert, wird sie aus den in der Blockchain enthaltenen Transaktionen erstellt. Neue in Blöcken enthaltene Transaktionen werden auf der State Database ausgeführt. Dies ermöglicht eine hohe Performance: Da die Datenbank im Arbeitsspeicher abgelegt werden kann, sind schnelle Schreib- und Lesevorgänge möglich [20].

Transaktionsfluss: Clients, Peer Nodes, Ordering Nodes In einem Fabric Netzwerk einigen sich die Unternehmen auf den zu nutzenden Chaincode für eine Anwendung. Dieser wird in der Blockchain gespeichert. Clients können über bestimmte Anwendungen Transaktionen über ihre Identität ausführen. Endorser Peer Nodes überprüfen die Rechte des Clients, die Validität der Transaktion und simulieren diese. Dazu führen sie die Transaktion auf der State Database aus, um die Datenänderungen zu erkennen. Diese werden jedoch noch nicht festgeschrieben. Anschließend werden die Transaktionen an eine Ordering Node geschickt. Diese sortiert die Transaktionen nach First-Come-First-Serve Prinzip in einen Block, welcher an die Committer Peer Nodes gesendet wird. Diese hängen den Block an die Blockchain an und führen die Datenänderungen (bereits simulierte Transaktionen) sequentiell auf der State Database durch. Dabei werden in Konflikt stehende Transaktionen erkannt und als invalide gekennzeichnet [20].

Development Die Entwicklung für Fabric erfolgt über Chaincode, welcher in Java oder Go geschrieben wird [70]. Um eine schnellere und komfortablere Entwicklung zu erlauben, wird das Framework Hyperledger Composer genutzt. Dieses wird im nächsten Abschnitt genauer betrachtet.

5.3.2 Hyperledger Composer

Hyperledger Composer (im folgenden nur noch Composer genannt) ist ein Framework für die Anwendungsentwicklung mit Fabric. Es bietet verschiedene Funktionen, welche das Implementieren von Blockchain-Applikationen beschleunigen. So ist es u. a. möglich Participants, Assets und Transaktionen zu modellieren, Zugriffsregeln festzulegen und daraus Chaincode sowie REST-API's zu generieren [71]. Dies alles wird im folgenden Kapiteln genauer erläutert. Es

ist zu bedenken, dass Composer kontinuierlich weiterentwickelt wird, weshalb die hier gemachten Angaben nicht mehr aktuell sein müssen [72]. Die Implementierung erfolgt mit Composer v0.16.2.

5.4 Entwicklungsumgebung

Als Entwicklungsumgebung wird eine Vagrant-Box, basierend auf Ubuntu 16.04 genutzt. Dabei handelt es sich um leichtgewichtige VM, mit welcher eine SSH-Verbindung hergestellt wird. Dies erlaubt das Arbeiten mit der VM über die Kommandozeile [73]. Für die Box wird ein Provision-Script geschrieben, welches die benötigten Komponenten installiert um Fabric und Composer nutzen zu können. Dieses kann ebenfalls genutzt werden, um beliebige Ubuntu-VM's einzurichten. Zu den Komponenten gehört beispielsweise das Composer Command Line Interface (CLI), mit welchen über die Kommandozeile u. a. Chaincode installiert werden kann [74]. Composer beinhaltet ebenfalls eine Fabric Blockchain-Konfiguration, mit welcher eine Blockchain mit einem Peer für Entwicklungszwecke gestartet werden kann. Im Kapitel 5.7 wird eine eigene Netzwerkkonfiguration erstellt.

5.5 Business Network Definition

Der erste Schritt der Implementierung ist die Entwicklung der Programmlogik. Diese wird über die Business Network Definition (BND) von Composer realisiert. Aus ihr wird letztendlich der Chaincode generiert, welcher bei den Peer-Nodes installiert wird. Weiterhin besteht die Möglichkeit eine REST-API zu generieren, mit welcher die Nutzer mit dem Chaincode interagieren können [75]. Die Anwendungslogik muss den folgenden Workflow erlauben: Maschinen, welche sich im Besitz von Unternehmen befinden, erkennen, dass sie eine Wartung benötigen. Sie führen mit einer von ihnen zugeteilten Identität eine Transaktion aus, welche einen Wartungsvertrag erstellt. Wartungsdienstleister können diesen annehmen. Sie melden sich beim Gerät an und führen die Wartung durch, wobei die Wartungsschritte geloggt werden. Nach ausgeführter Wartung schließt die Maschine den Vertrag. Bei allen Operationen ist zu bedenken, dass sie nur unter bestimmten Konditionen erfolgen dürfen. So kann beispielsweise ein Wartungsvertrag nur angenommen werden, wenn er nicht bereits von einem anderen Wartungsanbieter akzeptiert wurde. Weiterhin muss eine Preisabsprache zwischen Unternehmen und Wartungsdienstleister erfolgen können.

5.5.1 Anwendungslogik

Die BND besteht aus 4 Dateien. Ein Model-File modelliert die Participants, Assets, sowie Transaktionen. Weiterhin beschreibt ein JavaScript-File den auszuführenden Code (welcher Daten erstellt und/oder bearbeitet), beim Aufruf einer Transaktion. Ein ACL-File definiert welcher Participant welche Daten lesen/bearbeiten und löschen darf. Zuletzt gibt es noch ein

Query-File. Dieses definiert Queries, mit welchen Daten innerhalb der BND abgefragt werden können. Dies wird im Rahmen des Prototypen jedoch nicht genutzt. Stattdessen werden die Filter der REST-API genutzt, um Datenabfragen auszuführen (siehe Kapitel 5.6.1) [71]. Im Folgenden werden die einzelnen Files genauer erklärt, und Codebeispiele gezeigt.

Model-File Im Model File gilt es Teilnehmer am Netzwerk, verfügbare Asset-Typen, zu implementierende Transaktionen sowie Events zu definieren. Die Modellierung erfolgt in der Composer Modeling Language [76]

Die Participants des dezentralen Wartungsmarktes sind Unternehmen (*Company*), Wartungsdienstleister (*MaintenanceProvider*) sowie Maschinen (*Machine*). Die Maschinen besitzen einen *Owner*-Key, mit welchen Sie den Unternehmen zugeordnet werden können. Der Grund, warum die Maschinen als Participant definiert sind, ist die Identitätsverwaltung. In Composer kann jeden Participant eine eindeutige Identität zugewiesen werden, über welche Transaktionen ausgeführt werden [77]. So kann beispielsweise eindeutig zugeordnet werden, welche Maschine einen Wartungsvertrag erstellt. Ein Beispiel für die Definition einer Machine findet sich unter dem Listing 5.1.

```
participant Machine identified by machineId {
    o String machineId
    o String type
    o String model
    --> Company owner
}
```

Listing 5.1: Modellierung einer Machine. Keys können primitive Datentypen oder Referenzen zu anderen Assets sein.

In der BND bestehen drei Typen von Assets. Der *MachineStatus* gehört zu einer Maschine und gibt an, ob sie funktionstüchtig ist. Der *MaintenanceContract* enthält u. a. Informationen über den Wartungsgrund und den Status der Wartung. Letztendlich gibt es noch die *PaymentAgreement*. Diese dokumentiert die abgesprochene Auszahlung zwischen Unternehmen und Wartungsanbieter für einen bestimmten Wartungsvertrag. Listing 5.2 zeigt die Definition eines MaintenanceContract.

Ebenfalls müssen die zu implementierenden Transaktionen definiert werden, welche letztendlich Assets erstellen und bearbeiten. Die Transaktion *InitMaintenance* wird von einer Maschine aufgerufen, um einen Wartungsvertrag zu erstellen. *AcceptMaintenanceContract* wird von Wartungsanbietern aufgerufen, um einen Vertrag zu akzeptieren. Ebenfalls wird von ihnen die Transaktion *AddPerformedStep* genutzt, um ausgeführte Wartungsschritte zu loggen. Die *CloseContract*-Transaktion wird von der Maschine aufgerufen, nachdem der Wartungsanbieter beispielsweise einen Knopf an der Maschine drückt, um zu signalisieren, dass die Wartung erfolgt ist. Dabei erfolgt eine Überprüfung, ob der letzte erforderliche Wartungschritt erfolgt ist.

```
asset MaintenanceContract identified by maintenanceContractId {
    o String maintenanceContractId
    o String maintenanceReason
    o Boolean isAccepted
    o Boolean isClosed
    o String[] performedSteps
    o String requiredLastStep optional
    --> Machine owner
    --> MaintenanceProvider maintenanceProvider optional
}
```

Listing 5.2: Modellierung eines MaintenanceContract.

Letztendlich gibt es noch *CreatePaymentAgreement*, welche vom Unternehmen genutzt wird, um zu einen Vertrag einen Zahlungsvorschlag zu erstellen sowie *AcceptPaymentAgreement*, mit welcher der Wartungsanbieter diese akzeptieren kann. Listing 5.3 zeigt ein Beispiel für eine Transaktionsdefinition.

```
transaction AddPerformedStep {
    --> MaintenanceContract contract
    o String performedStep
}
```

Listing 5.3: Modellierung der AddPerformedStep-Transaktion. Die Keys sind in diesem Fall die mit der Transaktion übergebenen Parameter.

Zuletzt müssen noch die Events erwähnt werden. Diese werden gesendet, wenn bestimmte Trigger ausgelöst werden. Client-Applikation können diese abonnieren, um so beispielsweise bei Datenänderungen benachrichtigt zu werden [78]. So wird beispielsweise eine Website zum Annehmen von Wartungsverträgen automatisch aktualisiert, sobald ein neuer Vertrag in der Blockchain erstellt wird. Die Events welche bestehen, sind *NewContractCreated* und *ContractClosed*. Ersteres wird im Listing 5.4 definiert.

```
event NewContractCreated {
}
```

Listing 5.4: Modellierung eines Events, welches ausgelöst wird, sobald ein neuer Vertrag erstellt wird.

JavaScript-File Im Model-File Kapitel wurden Transaktionen und Events nur definiert. Das Verhalten der Transaktionen sowie die Trigger der Events werden im JavaScript-File festgelegt.

Aufgrund der Komplexität des Scripts wird nur beispielhaft die Implementation einer Transaktion sowie eines Events erläutert.

Das Listing 5.5 enthält den JavaScript-Code für die AcceptMaintenanceContract-Transaktion. Der übergebene Parameter *tx* enthält die Keys, welche im Model-File für die Transaktion definiert wurden. Bevor die Transaktion ausgeführt wird, wird überprüft ob der Contract bereits akzeptiert wurde und ob der Wartungsanbieter, welcher die Transaktion ausführt, die benötigte Erfahrung mit der zu wartenden Maschine hat (siehe Zeile 9-12). In Zeile 14-15 werden die neuen Werte für die Keys im MaintenanceContract gesetzt, damit der Vertrag als akzeptiert gilt. Die Änderungen muss nun noch in der State Database ausgeführt werden. Dazu wird in Zeile 18-21 die Asset-Registry aufgerufen, welche alle Assets enthält und das Update ausgeführt.

```
1  /**
2  * Accept the MaintenanceContract
3  * @param {biz.innovationcenter.maintenance.AcceptMaintenanceContract} tx The transaction
   ↪  instance.
4  * @transaction
5  */
6  function acceptMaintenanceContract(tx) {
7      var error = null;
8      //Check if the contract is already accepted
9      if (tx.maintenanceContract.isAccepted === false) {
10         //Check if the Participant has the required experience
11         if (tx.maintenanceContract.owner.type === getCurrentParticipant().experienceWith) {
12             //Set the new Contract Data for the contract parameter
13             tx.maintenanceContract.isAccepted = true;
14             tx.maintenanceContract.maintenanceProvider = getCurrentParticipant();
15
16             // Get the contract from the asset registry
17             return getAssetRegistry('biz.innovationcenter.maintenance.MaintenanceContract')
18                 .then(function (assetRegistry) {
19                     // Update the contract in the asset registry.
20                     return assetRegistry.update(tx.maintenanceContract);
21                 });
22         } else {
23             error = 'Error: Provider does not have required experience';
24         }
25     } else {
26         error = 'Error: Contract is already accepted';
27     }
28 }
```

Listing 5.5: JavaScript-Code für die AcceptMaintenanceContract-Transaktion.

Das Listing 5.6 zeigt einen Auszug aus der InitMaintenance-Transaktion. Nachdem in Zeile

2-3 ein Vertrag erstellt und der Asset-Registry hinzugefügt wurde, wird in Zeile 7-9 das im Model-File definierte Event *NewContractCreated* gesendet.

```
1  ...
2  .then(function (contractRegistry) {
3      return contractRegistry.add(contract);
4  })
5  .then(function() {
6      //Emit event
7      var factory = getFactory();
8      var newContractEvent = factory.newEvent(NAMESPACE, 'NewContractCreated');
9      emit(newContractEvent);
10 });
```

Listing 5.6: Auszug aus der InitMaintenance-Transaktion. Ein Event wird gesendet, nachdem ein neuer Vertrag erstellt wurde.

ACL-File Die Zugriffsregeln (ACL-Rules) bestimmen die Schreib- und Leserechte der einzelnen Participants. Eine Möglichkeit solche Regeln festzulegen, wurde bereits im vorherigen Abschnitt kurz gezeigt. In Zeile 12 vom JavaScript-Code des Listings 5.5 ile 12 wird überprüft, ob ein Anbieter die benötigte Erfahrung mit der zu wartenden Maschine hat. Es empfiehlt sich jedoch solche Regeln im ACL-File festzulegen, damit eine zentrale Stelle für diese existiert und die eventuell fehlende Berechtigung vor dem Ausführen des Codes erkannt wird. Ein weiteres Beispiel für eine ACL-Rule wäre, dass nur Wartungsanbieter, welche einen Wartungsvertrag angenommen haben, durchgeführte Wartungsschritte eintragen dürfen. Ein Beispiel dafür wäre Listing 5.7. In Zeile 6 wird überprüft, ob der im Vertrag angegebene Wartungsanbieter die Transaktion ausführt.

```
1  rule ProviderCanExecuteAddPerformedStepTransaction {
2      description: "Maintenance provider can add performed maintenance steps to his accepted
       ↪  contract"
3      participant(p): "biz.innovationcenter.maintenance.MaintenanceProvider"
4      operation: CREATE
5      resource(r): "biz.innovationcenter.maintenance.AddPerformedStep"
6      condition: (r.contract.maintenanceProvider.getIdentifier() == p.getIdentifier())
7      action: ALLOW
8  }
```

Listing 5.7: ACL-Rule, welche bestimmt, dass nur Wartungsanbieter, welche einen Vertrag akzeptiert haben, Wartungsschritte eintragen dürfen.

Den ACL-Rules fehlt in der genutzten Version von Composer ein nützliches Feature. Participants können Rechte nur auf gesamte Assets, allerdings nicht auf einzelne Keys dieser Assets

erhalten. Damit ein Wartungsanbieter die eben genannte Transaktion ausführen kann, benötigt er Update-Rechte für den Wartungsvertrag. Damit könnte er über eine Standard-Update-Transaktion den Status des Vertrags auf geschlossen stellen, obwohl nur eine Maschine die *CloseContract*-Transaktion ausführen kann. Das einzige was Participants davon abhält andere Teilnehmer dadurch zu schädigen, ist die Tatsache das jede ausgeführte Transaktion und die dazugehörige Identität in der Blockchain gespeichert wird [20].

5.5.2 Installation

Die Schritte für die Installation der BND erfolgen über das Composer CLI. Zunächst wird aus ihr ein Business Network Archive (BNA) generiert. Dieses wird bei den einzelnen Peers installiert, welche durch ein Connection Profile angegeben werden. Dieser Vorgang wird genauer im Kapitel 5.7 beschrieben. Für Entwicklungszwecke genügt zunächst die Installation bei den bereitgestellten Peer von Hyperleder Composer.

5.6 Client Applications

Nachdem die BND besteht, gilt es Client-Applikationen zu implementieren, um mit dieser zu interagieren. Dazu wird zuerst die REST-API erläutert, über welche die Anwendungen mit der Blockchain interagieren. Anschließend wird auf die Angular-Webanwendungen und die Gerätesimulation über eine Node.js Anwendung eingegangen. Um zu garantieren, dass die Client-Applications nicht von einer zentralen Instanz verwaltet werden, hostet jeder Teilnehmer diese sowie die REST-Server selbst.

5.6.1 REST-API

Composer bietet die Möglichkeit, eine REST-Schnittstelle aus der BND zu generieren. Sie erlaubt das Abfragen, Erstellen, Bearbeiten und Löschen von Assets sowie das Ausführen von Transaktionen. Dies erfolgt über GET oder POST Requests an bestimmte URLs. Beispiele dazu werden im folgenden Kapitel genannt. Zu der generierten API gehört ebenfalls eine Weboberfläche (siehe Abb. 5.1), welche einen Überblick über alle zur Verfügung stehenden REST-Aufrufe gibt sowie die Ausführung dieser erlaubt. An dieser Stelle muss erwähnt werden, dass die REST-API im Prototypen nur einen eingeloggten Nutzer unterstützt. Dieser wird im Source Code festgelegt. Dies führt dazu, dass für das Ausführen von Transaktionen mit unterschiedlichen Identitäten, unterschiedliche REST-Server genutzt werden müssen [79].

5.6.2 Webanwendungen

Maintenance-App Die Maintenance-App ist eine Angular-Webanwendung. Mit ihr können Wartungsanbieter noch nicht angenommene Wartungsverträge einsehen und akzeptieren. Weiterhin kann das Loggen von Wartungsschritten sowie das Überprüfen der Vertragsschließung

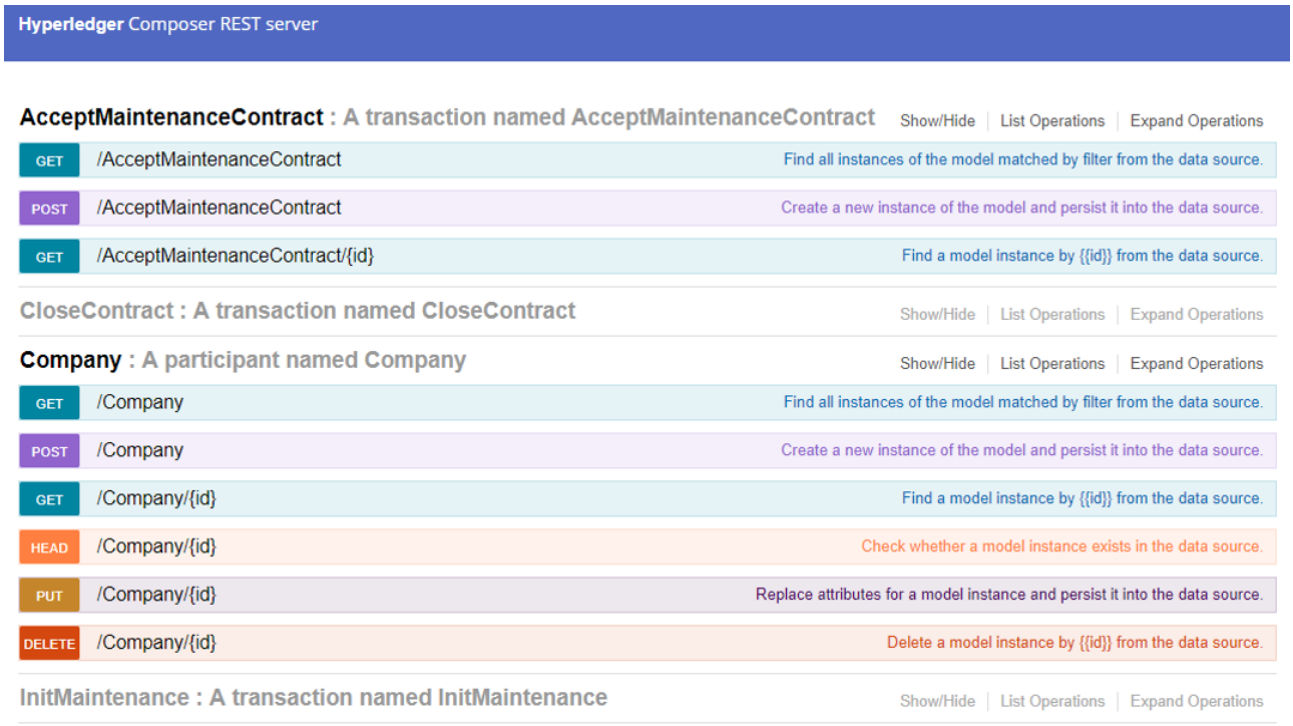

Abbildung 5.1: Generierte REST-API zu der BND. GET und POST Requests werden für Datenabfragen sowie das Ausführen von Transaktionen genutzt.

erfolgen. In der aktuellen Implementation ist der eingeloggte Nutzer durch den genutzten REST-Server definiert. Im Folgenden werden Codebeispiele zu den Abfragen und Erstellen von Assets sowie dem Abonnieren von Events gezeigt.

Listing 5.8 zeigt die Abfrage aller existierenden Wartungsverträge über einen GET-Request. Dieser wird über die in Zeile 2 angegebene URL an den REST-Server geschickt.

```
1 public fetchAllContracts() : Observable<any> {
2     let API_BASE = "http://10.40.94.180:12100/api/";
3     return this.http.get(API_BASE + "MaintenanceContract");
4 }
```

Listing 5.8: Abfrage aller existierenden Wartungsverträge.

Es ist ebenfalls möglich GET-Requests mit Filtern auszuführen. Im Listing 5.9 werden nur Wartungsverträge abgefragt, welche von den eingeloggten Wartungsanbieter akzeptiert, aber noch nicht geschlossen wurden. Die dazu in Zeile 5 entstandene URL wurde von der REST-Weboberfläche generiert.

Zuletzt wird ein Beispiel zum Ausführen einer Transaktion gezeigt. Wartungsanbieter können über die *AddPerformedStep*-Transaktion die durchgeführten Wartungsschritte loggen. Listing 5.10 zeigt den dafür auszuführenden POST-Request. In Zeile 2-6 wird ein JSON-Objekt erstellt, welches die auszuführende Transaktion sowie ihre Parameter festlegt. Dazu gehört der

```
public getAcceptedContracts() : Observable<any> {
    let API_BASE = "http://10.40.94.180:12100/api/";
    let PROVIDERNAME = "Aintenance";

    return this.http.get(API_BASE +
    ↪ "MaintenanceContract?filter=%7B%22where%22%3A%7B%22and%22%3A%5B%7B%22isAccepte
    ↪ d%22%3Atrue%7D%2C%7B%22isClosed%22%3Afalse%7D%2C%20%7B%22maintenanceProvide
    ↪ r%22%3A%22resource%3Abiz.innovationcenter.maintenance.MaintenanceProvider%23" +
    ↪ PROVIDERNAME + "%22%7D%5D%7D%7D");
}
```

Listing 5.9: Abfrage aller Wartungsverträge, welche vom eingeloggten Wartungsanbieter akzeptiert wurden und noch nicht geschlossen sind.

zu bearbeitende Vertrag sowie der ausgeführte Wartungschritt. Letztendlich wird das JSON-Objekt als Parameter der POST-Request übergeben und der im JavaScript-File der BND angegebene Code der Transaktion ausgeführt.

```
1 public addPerformedStep(operation : string, contractId : string) : Observable<any> {
2     const body = {
3         "$class": "biz.innovationcenter.maintenance.AddPerformedStep",
4         "contract": contractId,
5         "performedStep": operation
6     };
7     return this.http.post(API_BASE + 'AddPerformedStep', body);
8 }
```

Listing 5.10: POST-Request zum Hinzufügen von durchgeführten Wartungsschritten zum Wartungsvertrag.

Composer Playground Der Composer Playground ist eine in Composer enthaltene Webanwendung. Sie wird hauptsächtlich während der Entwicklung zum Testen der BND verwendet wird. Über den Playground können Participants sowie Identitäten für diese erstellt werden. Ebenfalls ist es möglich, Assets zu erstellen und zu bearbeiten. Was die Anwendung jedoch auch für den Endnutzer attraktiv macht, ist, dass sie einen Überblick über alle bestehenden Daten gibt. Weiterhin ist es möglich Transaktionen ausführen und die Historie aller durchgeführten Transaktionen einzusehen. Es ist jedoch nicht möglich Daten zu filtern. Im bestehenden Prototypen soll der Playground hauptsächlich von den Unternehmen eingesetzt werden, um die Daten der Maschinen und Wartungsverträge einzusehen. Weiterhin muss er in der aktuellen Version von den Unternehmen und Wartungsanbietern genutzt werden, um die Preisabsprachen in der Blockchain zu dokumentieren. Bei einer Weiterentwicklung sollten dafür ebenfalls Webanwendungen entwickelt werden.

5.6.3 XDK-Trigger

Um die Simulation von Wartungsgeräten zu realisieren, wird ein Bosch Cross Domain Kit (XDK) genutzt. Dabei handelt es sich um ein Gerät mit diversen Sensoren. So misst es u. a. Temperatur, Luftfeuchtigkeit sowie Beschleunigung (bzw. G-Kräfte). Auf dem XDK ist ein Programm installiert, welches die Sensordaten über USB per Serial Port überträgt. Wenn bestimmte Daten angeliefert werden, sollen Transaktionen ausgeführt werden, um Wartungsverträge zu erstellen und zu schließen. Dafür wird eine Node.js-Anwendung geschrieben.

Ein Wartungsvertrag wird erstellt, sobald die Luftfeuchtigkeit einen bestimmten Schwellwert überschreitet. Das Schließen von Wartungsverträgen erfolgt, sobald das Gerät auf den Kopf gedreht wird. Dies würde das Drücken eines Knopfes an der zu wartenden Maschine simulieren, wodurch der Wartungsanbieter die fertige Wartung signalisiert sowie die Schließung des Vertrags beantragt. Damit die Transaktionen mit der Identität einer Maschine ausgeführt werden, muss ein dementsprechend konfigurierter REST-Server existieren.

Listing 5.11 zeigt, wie mit dem Überschreiten der Luftfeuchtigkeitsschwellwerts umgegangen wird. In Zeile 7 wird die Funktion *createSmartContract* aufgerufen. Dieser führt die InitMaintenance-Transaktion über einen POST-Request aus. Dabei werden zufällige Werte für die Parameter *contractId*, *maintenanceReason* und *lastRequiredStep* übergeben.

```
1  function handleHumidityEvent() {
2      var HUMIDITY_TRIGGER = 70;
3      //If the humidity is too high, create a smart contract
4      if (jsonData.humidity >= HUMIDITY_TRIGGER) {
5          if (humidityFirstTimeExceeded) {
6              //Execute InitMaintenance-Transaction via POST-Request
7              console.log("HUMIDITY TOO HIGH! MY HAIR!!!");
8              createSmartContract();
9          }
10         humidityFirstTimeExceeded = false;
11     } else {
12         humidityFirstTimeExceeded = true;
13     }
14 }
```

Listing 5.11: Erstellen eines Wartungsvertrags bei der Überschreitung des Schwellwerts für die Luftfeuchtigkeit.

5.7 Netzwerkkonfiguration

Die Netzwerkkonfiguration besteht aus dem Erstellen einer Fabric-Netzwerk-Konfiguration sowie dem Konfigurieren von Composer zum Installieren der BND.

5.7.1 Fabric-Netzwerk-Konfiguration

Um das Blockchain-Netzwerk zu konfigurieren und zu starten, müssen Orderer Nodes, Peer Nodes, Certificate Authorities und Channel in mehreren Dateien definiert werden. Um diesen Prozess zu vereinfachen, wird das Tool *netcomposer*[80] genutzt. Dieses erstellt aus einem einzigen Konfigurationsfile alle benötigten Dateien, um ein Blockchain-Netzwerk zu starten.

In dem zu erstellenden Netzwerk soll es zwei Unternehmen (Eiva und Twimbee), zwei Wartungsanbieter (Repairr und Aintenance), sowie zwei Maschinen (Server_1 und Machine_1), welche zu den Unternehmen gehören, geben (siehe Abb. 5.2). Eine verkürzte Konfiguration dafür ist im Listing 5.12 angegeben. In Zeile 1 wird der genutzte Konsensmechanismus angegeben. In der aktuellen Implementation gibt es nur eine Ordering Node. Der Grund dafür wird genauer im Kapitel 5.8 erläutert. In Zeile 4 wird die für die State Database zu nutzende Datenbank definiert. Von Zeile 10-21 werden die Channels konfiguriert. Der Channel *mychannel* ist der öffentliche Channel, während *privatechannel* nur zwischen 2 Organisationen besteht.

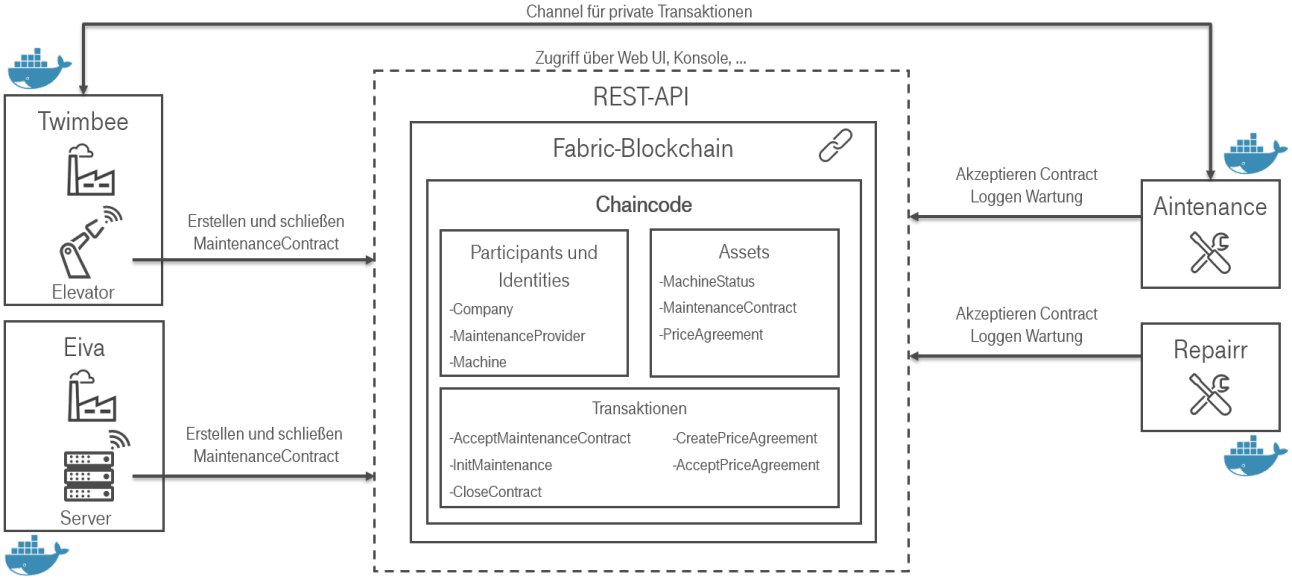

Abbildung 5.2: High-Level-Architektur des zu entstehenden Fabric-Netzwerks.

Aus diesem Konfigurationsfile werden verschiedene Dateien generiert, u. a. auch ein Docker-Compose-File. Fabric nutzt Docker-Container[1], um die Peer Nodes, Ordering Nodes, State Databases und Certificate Authorities zur Verfügung zu stellen. Das Docker-Compose-File wird genutzt, um die im Konfigurationsfile angegebenen Komponenten zu starten. Nach dem Start der Container muss die Kommandozeile genutzt werden, um die Channel zu starten und die Peers diesen hinzuzufügen. Dies erfolgt letztendlich über selbst erstellte Shell-Skripte.

[1]Docker-Container: Virtualisiertes Abbild, welches eine Applikation und alle Ressourcen zur Ausführung dieser beinhaltet [81].

```
orderer:
    type: "solo"
db:
    provider: "CouchDB"

organizations:      4
peersPerOrganization:   1
usersPerOrganization:   1

channels:
    - name: mychannel
        organizations:
        - organization: 1
        - organization: 2
        - organization: 3
        - organization: 4

    - name: privatechannel
        organizations:
        - organization: 1
        - organization: 2
```

Listing 5.12: Konfiguration des Fabric-Netzwerks (verkürzt).

5.7.2 Composer-Konfiguration

Um die BND zu installieren, müssen Connection Profiles erstellt werden. Über diese wird Composer u. a. mit den IP-Adressen der Peers bekannt gemacht. In der genutzten Version von Composer müssen für jede Organisation je Channel 2 Profile erstellt werden. Anschließend müssen über die Kommandozeile Identitäten beantragt werden, über welche die Installation der BND erfolgt. Ebenfalls werden Identitäten erstellt, welche die Admins der BND der einzelnen Organisationen darstellen. Letztendlich wird die *SetupDemo*-Transaktion ausgeführt, um die Beispieldaten zu erstellen. Im Zuge dessen werden auch Identitäten für die Maschinen erstellt. Der Prozess erfolgt über selbst erstellte Shell-Skripte. Der gesamte Vorgang ist im Detail genauer unter [82] beschrieben.

5.8 Konsensmechanismus

Kapitel 4.2 beschreibt verschiedene Konsensmechanismen. Für den Use-Case empfiehlt sich die Nutzung des PBFT. Dieser erlaubt bei der Teilnehmerzahl von 4 Nodes einen Transaktionsdurchsatz von mehr als 14000 TPS. Weiterhin werden bis zu 1/3 an unvertrauenswürdigen Nodes toleriert. Der PBFT ist allerdings noch nicht Out-of-the-box in Fabric implementiert. Dies soll in Zukunft jedoch noch erfolgen [83]. Für den Prototypen müsste also eine Implementation des PBFT manuell erfolgen. Aufgrund des Zeitaufwands erfolgt dies im Rahmen der Prototyp-Entwicklung nicht. Fabric implementiert aktuell nur einen Konsensmechanismus basierend auf Apache Kafka[2]. Dieser ist jedoch unsicher, da keine unvertrauenswürdigen Nodes toleriert werden. Genauere Informationen können [57] entnommen werden.

5.9 Showcase-Demo

Letztendlich wird für den Prototypen eine Demo entworfen. Diese demonstriert die Funktionsweise sowie diverse Workflows der Anwendung. Die Komponenten der Demo werden in Abbildung 5.3 visualisiert. Alle Komponenten können innerhalb der genannten Entwicklungsumgebung oder auf einer beliebigen Ubuntu-Umgebung laufen. Das Unternehmen Twimbee, welches einen Server besitzt, ist eine Node des Netzwerks. Auf dieser Node läuft ebenfalls ein REST-Server, über welchen der Server Wartungsverträge auf der (lokal laufenden) Blockchain erstellen kann. Weiterhin nutzt Twimbee den Playground, um Daten auf der öffentlichen sowie privaten Blockchain einzusehen. Auf den privaten Channel werden über den Playground ebenfalls Preisabsprachen dokumentiert. Der Wartungsanbieter Aintenance ist ähnlich aufgebaut. Der einzige Unterschied besteht darin, dass Aintenance anstatt eines Servers, eine lokale Webapplikation besitzt, welche mit seinen REST-Server kommuniziert. Eine noch nicht erwähnte Applikation ist der Hyperledger Blockchain Explorer [85]. Über ihn erfolgt die Einsicht

[2]Apache Kafka: Plattform zur Verarbeitung von Datenströmen [84].

und Visualisierung der Rohdaten auf der Blockchain. Die Anwendung kann ebenfalls bei jeden Teilnehmer selbst bereitgestellt sein.

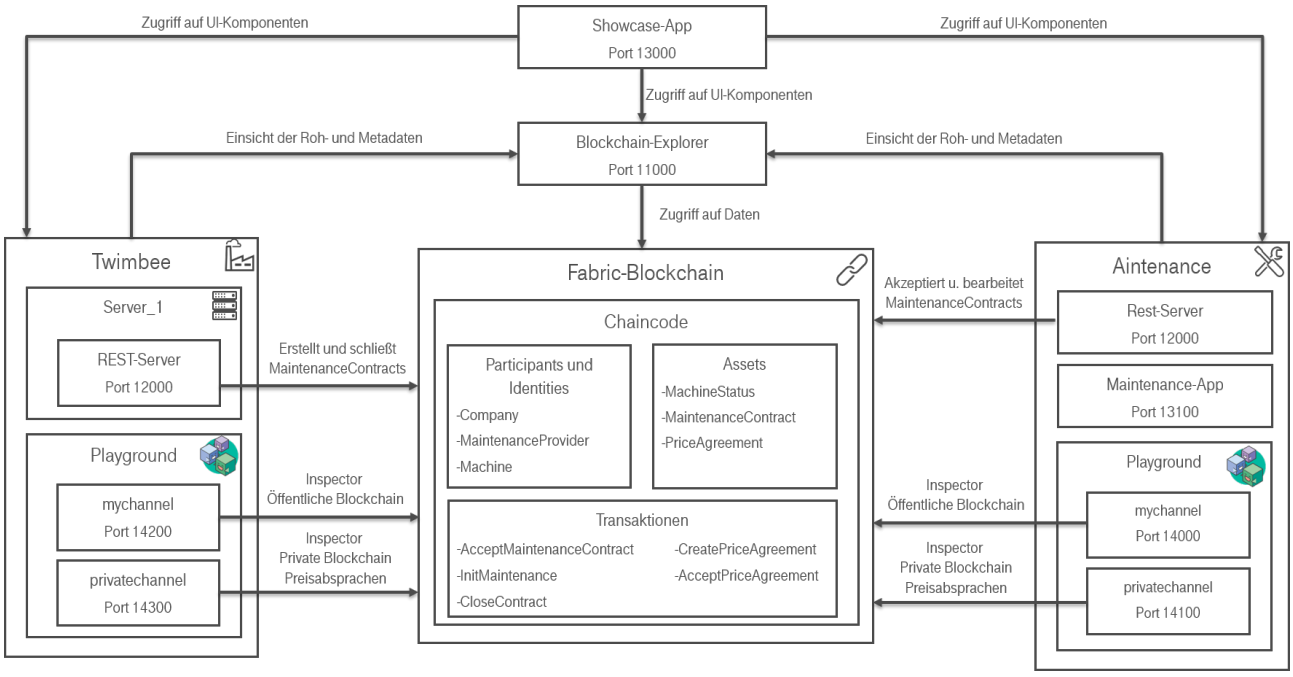

Abbildung 5.3: Architektur der Demo-Komponenten.

In der Demo wird der Workflow des Sequenzdiagramms der Abbildung 5.4 demonstriert. Es beschreibt den Workflow für die Wartung eines Geräts. Zunächst wird die Webapplikation der Wartungsanbieter geöffnet. Der Bosch XDK wird angehaucht, um den Schwellwert für die Luftfeuchtigkeit zu überschreiten. Anschließend wird vom ihm die *InitMaintenance*-Transaktion über den REST-Server aufgerufen und somit ein Wartungsvertrag erstellt. Dieser Wartungsvertrag erscheint daraufhin in der Webapplikation der Wartungsanbieter und wird angenommen. Danach werden verschiedene Wartungsschritte geloggt, inklusive des letzten geforderten Schrittes. Die eingetragenen Schritte werden in der Webapplikation sichtbar. Letztendlich wird der Bosch XDK auf den Kopf gedreht, wodurch die Schließung des Vertrags beantragt wird. Ist der letzte erforderliche Wartungsschritt in der Blockchain eingetragen, ist dies erfolgreich.

Anschließend wird kurz der Playground für die Dateneinsicht vorgestellt. In Zuge dessen wird gezeigt, dass es möglich ist, den Playground auf den öffentlichen und privaten Channel zu nutzen. Um die Funktion des privaten Channels zu zeigen, wird eine Preisabsprache von Twimbee erstellt. Wenn der Wartungsanbieter mit dieser einverstanden ist, kann er sie akzeptieren. Anschließend wird gezeigt, dass die Preisabsprache nicht im öffentlichen Channel vorkommt.

In der Demo erfolgt als letztes die Demonstration des Blockchain Explorer. Über ihn kann u. a. die Anzahl der bestehenden Blöcke und Peers sowie die Details der Transaktionen eingesehen werden.

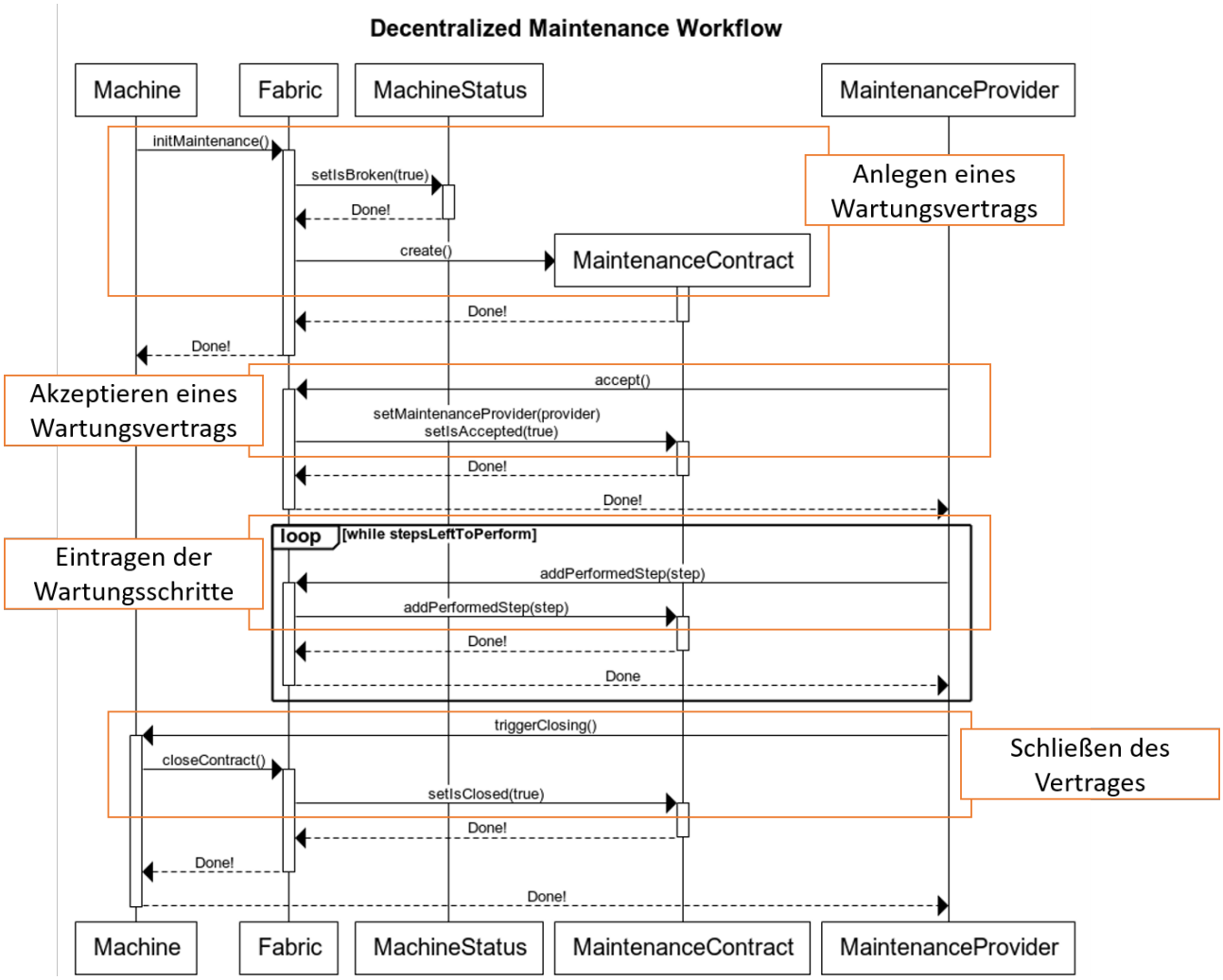

Abbildung 5.4: Workflow für die Wartung.

Kapitel 6

Fazit und Ausblick

Die Blockchain-Technologie hat das Potenzial, dezentrale Anwendungen zu ermöglichen, in welchen Transaktionen nicht manipuliert oder gelöscht werden können. Öffentliche Blockchains bringen jedoch noch Probleme bezüglich Skalierbarkeit und Privatsphäre mit sich. Permissioned Blockchains können diese Probleme zu Teilen lösen, nehmen dafür aber eine stärkere Zentralisierung und weniger sichere Konsensmechanismen in Kauf.

Auf Grundlage dieser Erkenntnisse wurde ein dezentraler Wartungsmarkt entwickelt. Die Implementation von diesen mit der Blockchain bringt verschiedene Vorteile gegenüber klassischen B2B-Anwendungen mit sich. Die Teilnehmer müssen nicht einer zentral verwaltenden Instanz vertrauen und können ihre Daten trotzdem an einer zentralen Stelle zur Verfügung stellen.

Der Prototyp des dezentralen Wartungsmarktes stellt ein Proof-of-Concept für diese da. Es gilt jedoch noch verschiedene Dinge zu bedenken, Limitationen zu untersuchen und Funktionen zu verbessern. So kann keine eindeutige Aussage zu der Skalierbarkeit und Performance getroffen werden. Der maximal mögliche Transaktionsdurchsatz von Hyperledger Fabric müsste mit leistungsstarken Computern untersucht werden, um eine Aussage darüber zu treffen, ob ein Wartungsmarkt mit mehreren tausend Teilnehmern und Maschinen bestehen kann. Diesbezüglich muss ebenfalls eine Analyse des PBFT erfolgen, wenn es mehr als 64 Nodes gibt. Ebenfalls besteht die Frage, ob es ausreichend ist, wenn 1/3 an unvertrauenswürdigen Teilnehmern toleriert werden. Deshalb ist es wichtig, dass weitere Analysen von Hyperledger Fabric in der Zukunft erfolgen. Letztendlich steht fest, dass die Performance mit der Anzahl an Nutzern sinkt und Anwendungen deswegen auf einen bestimmten Transaktionsdurchsatz bzw. auf eine bestimmte Anzahl an Peer-Nodes beschränkt sind.

Der entwickelte dezentrale Wartungsmarkt beinhaltet nur Features, welche für einen Proof-of-Concept benötigt werden. Er kann in diversen Bereichen erweitert werden. So können beispielsweise Remote-Support-Verträge eingeführt werden. Die Wartungsanbieter könnten, falls sie mit der Wartung keinen Erfolg haben, Hilfe über ihr Smartphone oder ihre Smart Glasses anfordern. Experten, welche Teilnehmer an der Blockchain sind, würden dann beispielsweise über ein Videogespräch den Support leisten. Ein weiteres interessantes Konzept ist ein Reputation

System. Heutige Bewertungssysteme haben das Problem, dass falsche Bewertungen abgegeben werden, beispielsweise um Nutzer zu schädigen. Die Bewertung der Wartungsanbieter könnte jedoch automatisch über in der Blockchain bestehende Daten, wie zum Beispiel die benötigte Zeit für eine Wartung, erfolgen. Solche Systeme werden ebenfalls in den Arbeiten von Carboni [86] und Dennis [87] vorgestellt.

Ebenfalls bringen Hyperledger Fabric und Composer selbst Probleme mit sich. Die Konfiguration eines Netzwerks ist mit sehr viel Aufwand verbunden. Um im aktuellen Prototyp neue Teilnehmer hinzuzufügen, muss eine neue Konfiguration erstellt werden, wodurch auch alle davon abhängigen Shell-Skripte bearbeitet werden müssen. Ebenfalls gibt es nur einen mitgelieferten nutzbaren Konsensmechanismus (Kafka). Andere, wie PBFT, müssten selbst implementiert werden, was zusätzlichen Aufwand bedeutet. Das fehlende Data Sharing zwischen Channeln kann ebenfalls problematisch sein. Es ist nicht möglich, Daten vom Public Channel mit dem Private Channel zu teilen. So können private Transaktionen keine Referenz zu einem im Public Channel bestehenden Asset herstellen.

Ein letzter Punkt, welcher in Bezug zu Blockchain-Anwendungen erwähnt werden muss, ist, dass die Blockchain keine Datenrichtigkeit garantiert. So sagen beispielsweise Wüst et al., dass "das Interface zwischen physischer und digitaler Welt" ein Problem darstellt [2]. So könnte beispielsweise beim Prototyp ein Wartungsanbieter durchgeführte Wartungsschritte dokumentieren, welche gar nicht erfolgt sind. Um dies zu lösen, könnte ein Sensor anhand verschiedener Daten überprüfen ob ein Wartungsschritt erfolgt ist. Dann muss allerdings diesen vertraut werden. Um Wartungsanbieter zu schädigen, könnte dieser von Unternehmen manipuliert sein. Letztendlich garantiert die Blockchain nur, dass eingetragene Daten bzw. ausgeführte Transaktionen nicht manipulier- und löschbar sind sowie bei allen Teilnehmern gleich vorkommen. Um falschen Input zu verhindern, müssen andere Lösungen gefunden werden.

Auch wenn es noch Fragen zu klären gilt, zeigt der entwickelte Wartungsmarkt die Machbarkeit von Blockchain-Anwendungen für B2B. Ob die Blockchain in Zukunft klassische B2B-Anwendungen ablöst, wird sich jedoch erst noch zeigen. Die Probleme der Technologie sowie der Nutzen von Permissioned Blockchains müssen weiterhin untersucht werden. Ebenfalls wichtig ist, dass mehr Blockchain-Anwendungen entwickelt und eingesetzt werden. So werden die sinnvollen Use-Cases ergründet und die Technologie besser verstanden. Nur so kann sich die Blockchain im B2B-Bereich etablieren.

Literatur

[1] Kari Korpela, Jukka Hallikas und Tomi Dahlberg. "Digital Supply Chain Transformation toward Blockchain Integration". In: 4. Jan. 2017. ISBN: 978-0-9981331-0-2. DOI: 10.24251/HICSS.2017.506. URL: http://scholarspace.manoa.hawaii.edu/handle/10125/41666 (besucht am 25. 12. 2017).

[2] Karl Wüst und Arthur Gervais. *Do You Need a Blockchain?* 375. 2017. URL: http://eprint.iacr.org/2017/375 (besucht am 08. 11. 2017).

[3] Michael Crosby. *BlockChain Technology: Beyond Bitcoin*. 2016.

[4] Vincent Gramoli. *On the Danger of Private Blockchains*. 2016.

[5] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008.

[6] Ethereum Team. *Ethereum White Paper*. ethereum, 27. Dez. 2017. URL: https://github.com/ethereum/wiki/wiki/White-Paper (besucht am 27. 12. 2017).

[7] Zibin Zheng et al. "Blockchain Challenges and Opportunities: A Survey". In: *International Journal of Web and Grid Services*. 14. Dez. 2017.

[8] Andreas M. Antonopoulos. *Mastering Bitcoin*. First edition. OCLC: ocn876351095. Sebastopol CA: O'Reilly, 2015. 272 S. ISBN: 978-1-4493-7404-4 978-1-4919-0260-8.

[9] Hyperledger Fabric Team. *Hyperledger Whitepaper*. 2016. URL: https://docs.google.com/document/d/1Z4M_qwILLRehPbVRUsJ3OF8Iir-gqS-ZYe7W-LE9gnE/edit?usp=embed_facebook (besucht am 08. 11. 2017).

[10] Melanie Swan. *Blockchain: Blueprint for a New Economy*. First edition. OCLC: ocn898924255. Beijing : Sebastopol, CA: O'Reilly, 2015. 130 S. ISBN: 978-1-4919-2049-7.

[11] Ameer Rosic. *What Is Hashing? Under The Hood Of Blockchain*. Juli 2017. URL: https://blockgeeks.com/guides/what-is-hashing/ (besucht am 29. 01. 2018).

[12] N. Kshetri. "Can Blockchain Strengthen the Internet of Things?" In: *IT Professional* 19.4 (2017), S. 68–72. ISSN: 1520-9202. DOI: 10.1109/MITP.2017.3051335.

[13] Hyperledger Composer Team. *Access Control Language - Hyperledger Composer*. URL: https://hyperledger.github.io/composer/unstable/reference/acl_language (besucht am 27. 12. 2017).

[14] Wikimedia Commons. *Public Key Signing*. 7 August 2006 (original upload date). URL: https://commons.wikimedia.org/wiki/File:Public_key_signing.svg (besucht am 29. 01. 2018).

[15] Rouse Margaret. *Nonce Definition*. URL: http://searchsecurity.techtarget.com/definition/nonce (besucht am 03. 02. 2018).

[16] Bitcoin Team. *Bitcoin Glossar*. URL: https://bitcoin.org/de/glossar (besucht am 27. 12. 2017).

[17] Etherscan. *Ethereum Network HashRate Growth Chart*. URL: https://etherscan.io/chart/hashrate (besucht am 27. 12. 2017).

[18] Etherscan. *Ethereum Average BlockTime Chart*. URL: https://etherscan.io/chart/blocktime (besucht am 28. 12. 2017).

[19] Bitcoinmining. *Learn about Bitcoin Mining Hardware*. URL: https://www.bitcoinmining.com/bitcoin-mining-hardware/ (besucht am 04. 01. 2018).

[20] Mattias Scherer. "Performance and Scalability of Blockchain Networks and Smart Contracts". 2017. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:umu:diva-136470 (besucht am 05. 01. 2018).

[21] BitcoinStats. *Bitcoin Propagation Data*. URL: http://bitcoinstats.com/network/propagation/ (besucht am 28. 12. 2017).

[22] Christian Decker und Roger Wattenhofer. "Information Propagation in the Bitcoin Network". In: *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference On*. IEEE, 2013, S. 1–10.

[23] H. Sukhwani et al. "Performance Modeling of PBFT Consensus Process for Permissioned Blockchain Network (Hyperledger Fabric)". In: *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*. 2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS). Sep. 2017, S. 253–255. DOI: 10.1109/SRDS.2017.36.

[24] Stefano De Angelis et al. "PBFT vs Proof-of-Authority: Applying the CAP Theorem to Permissioned Blockchain". In: Italian Conference on Cybersecurity. Unter Mitarb. von Stefano De Angelis et al. 2017. URL: https://eprints.soton.ac.uk/415083/ (besucht am 08. 11. 2017).

[25] Elyes Ben Hamida et al. "Blockchain for Enterprise: Overview, Opportunities and Challenges". In: *The Thirteenth International Conference on Wireless and Mobile Communications (ICWMC 2017)*. Nice, France, Juli 2017. URL: https://hal.archives-ouvertes.fr/hal-01591859 (besucht am 08. 11. 2017).

[26] Wenting Li et al. “Towards Scalable and Private Industrial Blockchains”. In: *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*. BCC ’17. New York, NY, USA: ACM, 2017, S. 9–14. ISBN: 978-1-4503-4974-1. DOI: 10.1145/3055518.3055531. URL: http://doi.acm.org/10.1145/3055518.3055531 (besucht am 08. 11. 2017).

[27] Don Tapscott und Alex Tapscott. *Die Blockchain-Revolution: Wie die Technologie hinter Bitcoin nicht nur das Finanzsystem, sondern die ganze Welt verändert*. 1. Aufl. Kulmbach: Plassen Verlag, 26. Okt. 2016. 440 S. ISBN: 978-3-86470-388-1.

[28] Amy Castor. *An Ethereum Voting Scheme That Doesn’t Give Away Your Vote*. 6. Apr. 2017. URL: https://www.coindesk.com/voting-scheme-ethereum-doesnt-give-away-vote/ (besucht am 04. 01. 2018).

[29] Winfried Krieger. *Supply Chain Management - Definition*. URL: http://wirtschaftslexikon.gabler.de/Definition/supply-chain-management-scm.html (besucht am 04. 01. 2018).

[30] Danny Dig und Ralph Johnson. “How Do APIs Evolve? A Story of Refactoring”. In: *Journal of Software: Evolution and Process* 18.2 (2006), S. 83–107.

[31] Future Flux Festival. *Blockchain Bikes*. URL: http://futurefluxfestival.nl/en/program/blockchain-bikes/ (besucht am 05. 01. 2018).

[32] Andreas Fischer. *Das IoT in Der Blockchain*. URL: https://www.com-magazin.de/praxis/internet-dinge/iot-in-blockchain-1228562.html (besucht am 05. 01. 2018).

[33] Steven Buchko. *How Long Do Bitcoin Transactions Take?* 12. Dez. 2017. URL: https://coincentral.com/how-long-do-bitcoin-transfers-take/ (besucht am 08. 01. 2018).

[34] van der Meulen Rob. *Gartner Says 8.4 Billion Connected "Things"Will Be in Use in 2017*. 7. Feb. 2017. URL: https://www.gartner.com/newsroom/id/3598917 (besucht am 02. 02. 2018).

[35] Digiconomist. *Bitcoin Energy Consumption Index*. URL: https://digiconomist.net/bitcoin-energy-consumption (besucht am 08. 01. 2018).

[36] Kasey Panetta. *Top Trends in the Gartner Hype Cycle for Emerging Technologies, 2017*. 15. Aug. 2017. URL: https://www.gartner.com/smarterwithgartner/top-trends-in-the-gartner-hype-cycle-for-emerging-technologies-2017/ (besucht am 30. 01. 2018).

[37] Ann Forni Amy. *Gartner’s 2016 Hype Cycle for Emerging Technologies Identifies Three Key Trends That Organizations Must Track to Gain Competitive Advantage*. 16. Aug. 2016. URL: https://www.gartner.com/newsroom/id/3412017 (besucht am 30. 01. 2018).

[38] Google. *Blockchain - Google Trends*. 30. Jan. 18. URL: /trends/explore (besucht am 30. 01. 2018).

[39] BlockchainHub. *Blockchains & Distributed Ledger Technologies*. URL: https://blockchainhub.net/blockchains-and-distributed-ledger-technologies-in-general/ (besucht am 09. 01. 2018).

[40] Danny de Boer. *AXA nutzt Ethereum-Blockchain für Flugversicherung*. 14. Sep. 2017. URL: https://www.btc-echo.de/axa-nutzt-ethereum-blockchain-fuer-flugversicherung/ (besucht am 30. 01. 2018).

[41] Marko Vukolić. "Rethinking Permissioned Blockchains". In: ACM Press, 2017, S. 3–7. ISBN: 978-1-4503-4974-1. DOI: 10.1145/3055518.3055526. URL: http://dl.acm.org/citation.cfm?doid=3055518.3055526 (besucht am 08. 11. 2017).

[42] Sirajd Raval. *Decentralized Applications: Harnessing Bitcoin's Blockchain Technology*. S. 33 ff. O'Reilly Media, Inc., 2016.

[43] Ido Kaiser. *A Decentralized Private Marketplace: DRAFT 0.1*. März 2017.

[44] Jagdeep Sidhu. "Syscoin: A Peer-to-Peer Electronic Cash System with Blockchain-Based Services for E-Business". In: *Computer Communication and Networks (ICCCN), 2017 26th International Conference On*. IEEE, 2017, S. 1–6.

[45] John Soldatos. *What Does Blockchain Technology Have to Do with Enterprise Maintenance Activities?* Juni 2017. URL: https://www.solufy.com/blog/what-does-blockchain-technology-have-to-do-with-enterprise-maintenance-activities (besucht am 08. 01. 2018).

[46] Patrick Götze. *Lufthansa Industry Solutions - Mit Blockchain Zu Mehr Transparenz in Der Luftfahrt*. URL: https://www.lufthansa-industry-solutions.com/de-de/loesungen-produkte/luftfahrt/mit-blockchain-zu-mehr-transparenz-in-der-luftfahrt/ (besucht am 09. 01. 2018).

[47] Victor Estuardo Araujo Soto. "Performance Evaluation of Scalable and Distributed IoT Platforms for Smart Regions". S. 15 ff. 2017.

[48] Bitcoin Team. *Scalability - Bitcoin Wiki*. URL: https://en.bitcoin.it/wiki/Scalability (besucht am 08. 01. 2018).

[49] Bitcoin.com Team. *Bitcoin Transaction Size*. URL: https://charts.bitcoin.com/chart/transaction-size (besucht am 31. 01. 2018).

[50] Blockchain (Unternehmen). *Unbestätigte Transaktionen - Bitcoin Blockchain Charts*. URL: https://blockchain.info/de/unconfirmed-transactions (besucht am 10. 01. 2018).

[51] Yonatan Sompolinsky und Aviv Zohar. *Accelerating Bitcoin's Transaction Processing. Fast Money Grows on Trees, Not Chains*. 881. 2013. URL: http://eprint.iacr.org/2013/881 (besucht am 12. 01. 2018).

[52] Vitalik Buterin. *Toward a 12-Second Block Time*. 11. Juli 2014. URL: https://blog.ethereum.org/2014/07/11/toward-a-12-second-block-time/ (besucht am 12. 01. 2018).

[53] Hyperledger Fabric Team. *Hyperledger Fabric - Releases*. 18. Jan. 2018. URL: https://github.com/hyperledger/fabric (besucht am 18. 01. 2018).

[54] Chris Mitchell. *Trusted Computing*. IET, 2005. 336 S. ISBN: 978-0-86341-525-8.

[55] Lin Chen et al. "On Security Analysis of Proof-of-Elapsed-Time (PoET)". In: *International Symposium on Stabilization, Safety, and Security of Distributed Systems*. Springer, 2017, S. 282–297.

[56] Gideon Greenspan. *MultiChain Private Blockchain - White Paper*. 2015. URL: https://www.multichain.com/download/MultiChain-White-Paper.pdf (besucht am 17. 01. 2018).

[57] Christian Cachin und Marko Vukolić. "Blockchain Consensus Protocols in the Wild". In: (6. Juli 2017). arXiv: 1707.01873 [cs]. URL: http://arxiv.org/abs/1707.01873 (besucht am 08. 11. 2017).

[58] Marko Vukolić. "The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication". In: *International Workshop on Open Problems in Network Security*. Springer, 2015, S. 112–125.

[59] Kyle Croman et al. "On Scaling Decentralized Blockchains". In: *Financial Cryptography and Data Security*. International Conference on Financial Cryptography and Data Security. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 26. Feb. 2016, S. 106–125. ISBN: 978-3-662-53356-7 978-3-662-53357-4. DOI: 10.1007/978-3-662-53357-4_8. URL: https://link.springer.com/chapter/10.1007/978-3-662-53357-4_8 (besucht am 27. 12. 2017).

[60] Tendermint Team. *Tendermint - Github Repository*. 17. Jan. 2018. URL: https://github.com/tendermint/tendermint (besucht am 17. 01. 2018).

[61] Jae Kwon. "Tendermint: Consensus without Mining". In: *Draft v. 0.6, fall* (2014).

[62] Ethan Buchman. "Tendermint: Byzantine Fault Tolerance in the Age of Blockchains". S. 18 ff. PhD Thesis. 2016.

[63] Quorum Team. *Transaction Processing - Quorum Wiki*. 17. Jan. 2018. URL: https://github.com/jpmorganchase/quorum (besucht am 17. 01. 2018).

[64] Quorum Team. *QuorumChain Consensus*. 31. Jan. 2018. URL: https://github.com/jpmorganchase/quorum (besucht am 31. 01. 2018).

[65] Blockchain (Unternehmen). *Blockchain Size - Bitcoin Blockchain Charts*. URL: https://blockchain.info/blocks-size (besucht am 17. 01. 2018).

[66] Hyperledger Fabric Team. *Support - Hyperledger Fabric Documentation*. URL: https://hyperledger-fabric.readthedocs.io/en/release/questions.html (besucht am 18. 01. 2018).

[67] Hyperledger Burrow Team. *Hyperledger Burrow - GitHub Repository*. 17. Jan. 2018. URL: https://github.com/hyperledger/burrow (besucht am 18. 01. 2018).

[68] Hyperledger Fabric Team. *Chaincode - Hyperledger Fabric Documentation*. URL: http://hyperledger-fabric.readthedocs.io/en/release/chaincode.html (besucht am 18. 01. 2018).

[69] Hyperledger Fabric Team. *CA - Hyperledger Fabric Documentation*. URL: http://hyperledger-fabric-ca.readthedocs.io/en/latest/users-guide.html#overview (besucht am 18. 01. 2018).

[70] Hyperledger Fabric Team. *SDKs - Hyperledger Fabric Documentation*. URL: https://hyperledger-fabric.readthedocs.io/en/release/fabric-sdks.html (besucht am 18. 01. 2018).

[71] Hyperledger Composer Team. *Introduction - Hyperledger Composer*. URL: https://hyperledger.github.io/composer/introduction/introduction.html (besucht am 08. 11. 2017).

[72] Hyperledger Composer Team. *Hyperledger Composer - Releases*. 19. Jan. 2018. URL: https://github.com/hyperledger/composer (besucht am 19. 01. 2018).

[73] Vagrant Team. *Vagrant by HashiCorp*. URL: https://www.vagrantup.com/index.html (besucht am 20. 01. 2018).

[74] Hyperledger Composer Team. *Development Environment - Hyperledger Composer*. URL: https://hyperledger.github.io/composer/installing/development-tools.html (besucht am 20. 01. 2018).

[75] Hyperledger Composer Team. *Developer Tutorial - Hyperledger Composer*. URL: https://hyperledger.github.io/composer/tutorials/developer-tutorial (besucht am 19. 01. 2018).

[76] Hyperledger Composer Team. *Modeling Language - Hyperledger Composer*. URL: https://hyperledger.github.io/composer/reference/cto_language.html (besucht am 19. 01. 2018).

[77] Hyperledger Composer Team. *Participants and Identities | Hyperledger Composer*. URL: https://hyperledger.github.io/composer/managing/participantsandidentities (besucht am 19. 01. 2018).

[78] Hyperledger Composer Team. *Emitting Events - Hyperledger Composer*. URL: https://hyperledger.github.io/composer/business-network/publishing-events.html (besucht am 31. 01. 2018).

[79] Hyperledger Composer Team. *REST API - Hyperledger Composer*. URL: https://hyperledger.github.io/composer/integrating/getting-started-rest-api.html (besucht am 20. 01. 2018).

[80] IBM Silvergate Team. *Netcomposer - Github Repository*. 5. Jan. 2018. URL: https://github.com/ibm-silvergate/netcomposer (besucht am 20. 01. 2018).

[81] Docker Team. *What Is a Container*. 29. Jan. 2017. URL: https://www.docker.com/what-container (besucht am 21. 01. 2018).

[82] Hyperledger Composer Team. *Multi Org Deployment - Hyperledger Composer*. URL: https://hyperledger.github.io/composer/tutorials/deploy-to-fabric-multi-org.html (besucht am 21. 01. 2018).

[83] Hyperledger Fabric Team. *Pluggable Consensus Implementations - Hyperledger Fabric Documentation*. URL: http://hyperledgerdocs.readthedocs.io/en/latest/pluggableos.html?highlight=Ordering%20Service (besucht am 20. 01. 2018).

[84] Apache. *Introduction - Apache Kafka*. URL: https://kafka.apache.org/intro (besucht am 02. 02. 2018).

[85] Hyperledger Blockchain Explorer Team. *Hyperledger Blockchain Explorer - GitHub Repository*. 21. Jan. 2018. URL: https://github.com/hyperledger/blockchain-explorer (besucht am 21. 01. 2018).

[86] Davide Carboni. "Feedback Based Reputation on Top of the Bitcoin Blockchain". In: (5. Feb. 2015). arXiv: 1502.01504 [cs]. URL: http://arxiv.org/abs/1502.01504 (besucht am 09. 01. 2018).

[87] Richard Dennis und Gareth Owen. "Rep on the Block: A next Generation Reputation System Based on the Blockchain". In: *Internet Technology and Secured Transactions (ICITST), 2015 10th International Conference For*. IEEE, 2015, S. 131–138.