Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Fachbereich Informatik
Computer Science Department

Abschlussarbeit

im Masterstudiengang Informatik

Data Provenance mit Ethereum und Untersuchung von Performanzaspekten

von Christopher Arera
beim Deutschen Zentrum für Luft- und Raumfahrt

Erstbetreuerin:	**Prof. Dr. Kerstin Lemke-Rust**
Zweitbetreuerin:	**Prof. Dr. Simone Bürsner**
Externer Betreuer:	**Andreas Schreiber**

Eingereicht am: **07. Juni 2024**

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt bzw. nicht veröffentlicht.

Sankt Augustin, den 07. Juni 2024

Christopher Arera

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Kontext und Motivation

Beim Deutschen Zentrum für Luft- und Raumfahrt (DLR) gibt es unterschiedliche Prozesse, welche reproduzierbar und sicher vor Manipulationen dokumentiert werden müssen. Ein Beispiel für einen solchen Prozess ist die Entwicklung von Flugzeugteilen. Deren Entwurfsdaten müssen vor dem Einsatz im Flug von einer Zertifizierungsbehörde geprüft werden. Das DLR speichert diese Daten gemäß dem PROV Data Model des W3C, welches neben der Entstehung durch bspw. zugehörige Prozesse und Methoden auch Informationen über die Herkunft erfasst [38].
Allgemein ist dieses Konzept als „Data Provenance“ bekannt und hat das Ziel, die Entstehung der Daten über die zusätzlichen Provenance-Informationen reproduzierbar und somit überprüfbar zu dokumentieren, wodurch die Vertrauenswürdigkeit der Daten gewährleistet werden soll. In einem gerichteten azyklischen Graphen (DAG) können die daraus entstehenden Datensätze miteinander verbunden werden, so dass die gesamte Historie der Daten nachvollzogen werden kann.
Um die dadurch gewonnene Vertrauenswürdigkeit auch während der Übertragung an außenstehende Institutionen zu erhalten, soll eine Möglichkeit gefunden werden, die Daten auf digitalem Weg integritätsgeschützt zu übertragen. Eine Lösung für die Übertragung kann der Einsatz von Blockchain-Technologien darstellen, da diese über ihren dezentralen Charakter und die Art der Datenspeicherung eine ausreichende Integrität gewährleisten.

1.2 Problemstellung und Forschungsfragen

Als ein Nachteil beim Einsatz von Blockchains, wird in der Literatur oft die Performanz benannt. So werden die Engpässe einiger Implementierungen in Fan et al. [34] identifiziert. Auch andere Arbeiten haben Performance Tests mit unterschiedlichen Blockchains durchgeführt und diese miteinander verglichen [15], [55]. Meist wird für Ethereum dabei der Konsensalgorithmus Proof-of-Work (PoW) gewählt. Daher stellt sich die Frage, wie die Performance im Hinblick auf das Speichern von Provenance-Graphen unter der Verwendung der Konsensalgorithmen Proof-of-Stake (PoS) und Proof-of-Authority (PoA) ausfällt und welche Methoden und Kriterien sich für die Messungen eignen.

RQ1: *Wie verhält sich die Performance im Hinblick auf das Speichern von Provenance-Graphen unter der Verwendung der Konsensalgorithmen Proof-of-Stake (PoS) und Proof-of-Authority (PoA)?*

Weiterhin werden in Ethereum für Transaktionen Gebühren erhoben. Diese werden in Gas angegeben und sind von der Anzahl der auszuführenden Operationen auf der Blockchain, sowie der zu speichernden Datenmenge abhängig. Um Transaktionen in der Blockchain zu speichern, werden diese in Blöcken gesammelt. Für jeden Block ist ein Gas Limit definiert, welches der summierte Gas Bedarf der enthaltenen Transaktionen nicht überschreiten kann. Auf diese Weise wird indirekt eine Grenze für speicherbare Datenmengen gesetzt. Da diese Grenze beim Speichern von Provenance-Graphen relativ schnell erreicht ist, muss sie für den Anwendungszweck entsprechend angepasst werden. Im Zusammenhang mit den unterschiedlichen Konsensalgorithmen, soll der Einfluss unterschiedlicher Konfigurationen auf die Performance untersucht werden. Von Interesse ist auch die Anzahl der Knoten im Netz, da diese je nach aufgesetztem Netz variieren können.

RQ2: *Welchen Einfluss haben unterschiedliche Konfigurationen des Gas Limits und die Anzahl der Knoten auf die Performanz lokaler PoS und PoA Netze?*

Ebenfalls interessant sind die Konfigurationen der öffentlichen Ethereum Blockchain und der permissioned Blockchain Bloxberg [5]. Das Ethereum-Mainnet zeichnet sich dadurch aus, dass es das größte öffentliche Ethereum-Netz ist, welches den Konsensalgorithmus PoS verwendet. Bloxberg hingegen ist laut eigener Aussage das größte wissenschaftliche PoA Netz, welches von Research Organisationen betrieben wird. Beide Netze sind potenziell zum Speichern von Provenance-Graphen denkbar. Jedoch werden für das Mainnet finanzielle Ressourcen und für Bloxberg der Zugang benötigt.

RQ3: *Bis zu welcher Datenmenge ist es möglich Provenance-Graphen im Ethereum-Mainnet und in der Blockchain Bloxberg zu speichern?*

Beschriebene Vorgehensmodelle zur Konzeption von Performance-Tests orientieren sich teilweise an Anwendungsszenarien und deren Anforderungen (vgl. [66]). Diese Voraussetzungen sind für die Abschlussarbeit nicht gegeben, weshalb das Vorgehen bei der Konzeption angepasst werden muss. Außerdem müssen geeignete Kriterien gefunden werden, um die unterschiedlichen Netz-Konfigurationen hinsichtlich der Performance miteinander vergleichen zu können.

RQ4: *Welche Methoden und Kriterien eignen sich, um die Performance beim Speichern eines Provenance-Graphen zu messen?*

Um die Performance Messungen durchführen zu können, werden sowohl Testdaten, als auch Speicherverfahren benötigt. Auf https://openprovenance.org/ werden zwar einige Datensätze zur Verfügung gestellt, jedoch müsste die Größe der Graphen für die Tests händisch angepasst werden. Daher sollen in Vorbereitung auf die Performance Tests Möglichkeiten zur Erzeugung synthetischer Provenance-Graphen recherchiert werden.

RQ5: *Welche Möglichkeiten gibt es synthetische Provenance-Graphen parametrisiert zu erzeugen?*

Um die Testdaten zu speichern, steht das Verfahren von Kocadag [45] zur Verfügung. Dieses basiert jedoch auf Hashwerten, wodurch sich die zu speichernde Datenmenge trotz größerer Testdaten nicht verändert. Da sich das Verfahren daher nicht eignet die Fragestel-

lungen der Arbeit zu untersuchen, wird mindestens ein weiteres Verfahren zum Speichern von Testdaten benötigt.

RQ6: *Welche Speicherverfahren eignen sich, um das Performanzverhalten in Abhängigkeit zur Graphgröße zu untersuchen?*

1.3 Ziele

Um die Performanz messen zu können, soll neben dem Verfahren von Kocadag [45], mindestens ein weiteres Verfahren zum Speichern eines Provenance-Graphen ausgewählt werden. Die Verfahren sollen genutzt werden, um die Performanzaspekte besonders im Hinblick auf unterschiedliche Datenmengen zu untersuchen. Im Idealfall sollte sich während der Tests ein Skalierungsfaktor herauskristallisieren, welcher die Performance in Abhängigkeit zur Graphgröße beschreibt. So sollen die Anwendungsmöglichkeiten der Blockchain als Speicher für Provenance-Daten abgeschätzt werden können. Im Zuge dessen soll außerdem die maximal speicherbare Größe innerhalb von Ethereum festgestellt werden.
Das im ersten Schritt ausgewählte Verfahren soll prototypisch in Python implementiert werden. Dazu sollen der Provenance-Graph im Dateiformat von der Anwendung eingelesen und in der Blockchain Ethereum gespeichert werden. Ein in Ethereum gespeicherter Graph soll anschließend wieder ausgelesen und verifiziert werden können.
Es soll ein Konzept erarbeitet werden, um die Performanz der Speicherverfahren während der Interaktion mit der Blockchain anhand ausgewählter Kriterien zu messen. In das Konzept sollen unterschiedliche Konfigurationen des privaten Netzes miteinbezogen werden. Von Interesse sind auch die Konfigurationen der öffentlichen Ethereum Blockchain, als auch von der permissioned Blockchain Bloxberg [5]. Die Grenzen beider Systeme sollen im Hinblick auf die Speichermöglichkeit festgestellt werden.
Für das Messen der Performanz werden unterschiedlich große Provenance-Graphen benötigt. Daher sollen Ansätze recherchiert werden, um die Graphen nach definierten Größen generieren zu können und ein entsprechender Generator implementiert werden.

Annahmen:

- Ein Verfahren zum Speichern der vollständigen Provenance-Graphen kann aus der Literaturrecherche von Kocadag [45] entnommen und der off-chain Anteil in Python implementiert werden. Der on-chain Anteil wird mit der Sprache Solidity umgesetzt. Die Kommunikation zwischen dem Client und der Blockchain, ist mit Hilfe einer Python-Bibliothek möglich.

- Es können bestehende Verfahren zur Generierung synthetischer Graphen recherchiert werden. Auf deren Basis lässt sich eine eigene Implementierung umsetzten, welche Testgraphen nach vorgegebener Größe erzeugen kann.

- Die Kriterien zum Messen der Performance können aus anderen wissenschaftlichen Arbeiten übernommen werden. Das Konzept zur Durchführung orientiert sich an dem üblicher Performance Tests mit konkretem Anwendungsszenario.

- Nach Abschluss der Tests lässt sich ein Skalierungsfaktor aus den Ergebnissen ableiten. Dieser hängt in erster Linie von der konfigurierten Blockgröße ab. Es gibt einen linearen

Zusammenhang zwischen der Datenmenge, welche in einem Block übertragen werden kann und der Abnahme der Performance.

- Das maximale Limit in Bezug auf die Datenmenge einer einzelnen Transaktion liegt im Megabyte-Bereich. Auch falls theoretisch größere Dateien über das Gas Limit speicherbar wären, ist dies zu ineffizient.
- Die maximal speicherbaren Datenmengen pro Transaktion hängt vom Gas Limit und den erforderlichen Gebühren ab.
- Da bei Bloxberg das Gas Limit bei 10 Mio. liegt, können maximal etwa 15 kB an Daten pro Block gespeichert werden.
- Da bei der öffentlichen Ethereum Blockchain das Gas Limit bei 30 Mio. liegt, können maximal etwa 45 kB an Daten pro Block gespeichert werden.

1.4 Aufbau der Arbeit

Die Arbeit ist in fünf wesentliche Teile untergliedert. Nach der Einleitung geht es im zweiten Kapitel um den Stand der Forschung. In diesem werden die Methoden und das Vorgehen bei der Literaturrecherche beschrieben und die vorliegende Arbeit von anderen abgegrenzt.
Im dritten Kapitel werden die Grundlagen für das Konzept und die Implementierung vorgestellt. Diese sollen ein Grundverständnis für die Thematik schaffen, um die später in Kapitel vier und fünf verwendeten Methoden und das Vorgehen zur Umsetzung nachvollziehen zu können.
Im vierten Kapitel wird das entwickelte Konzept vorgestellt, welches die Planung umfasst, um die Ziele der Arbeit zu verwirklichen. Zu Beginn wird ein Überblick über die interessantesten recherchierten Verfahren zur Erzeugung von Provenance-Graphen geliefert. Anschließend folgen Anforderungen an den zu implementierenden Generator, welcher in Kapitel 5 umgesetzt werden soll. Basierend auf den Anforderungen wird das theoretische Vorgehen zur Erzeugung der Graphen beschrieben und eine Architektur für die spätere Umsetzung geplant. Anschließend erfolgt die Auswahl des Speicherverfahrens, welches zum Einspielen der Provenance-Graphen genutzt werden soll. Im Anschluss erfolgt ein kurzer Überblick über unterschiedliche Möglichkeiten die Performance für Blockchain-Systeme zu messen und es werden geeignete Metriken ausgewählt. Als nächstes folgt die Planung der Performance-Tests, welche in Anlehnung an einen Testprozess durchgeführt wird. Im wesentlichentlichen dienen die Schritte der Planung der Testdaten, der Workloads, der Konfiguration der Netze sowie vorbereitender Schritt zum Aufbau der Testumgebung.
Im fünften Kapitel werden die Überlegungen aus Kapitel vier prototypisch umgesetzt. Dies umfasst die Implementierung des ausgewählten Speicherverfahrens, des Datengenerators und die Erzeugung der Testgraphen mittels diesem. Außerdem wird der Aufbau des Testnetzes beschrieben sowie die verwendeten Skripte, um die Durchführung der Performance-Tests zu unterstützen. Im sechsten Kapitel werden die erzielten Ergebnisse aus der Arbeit ausgewertet und letzte Fragestellungen beantwortet. Dies umfasst insbesondere die Analyse der Testergebnisse im Hinblick auf die ausgewählten Metriken. Abschließend folgt ein Fazit zusammen mit dem Ausblick.

Kapitel 2

Stand der Forschung

Um den Stand der Forschung zu erfassen und darauf basierend die Forschungsfragen (RQ1-RQ6) bearbeiten zu können, wird eine systematische Literaturrecherche durchgeführt. Das Vorgehen orientiert sich dabei an der Methode von Kitchenham [44]. Ziel ist es die relevante Literatur möglichst vollständig zu sichten und den Prozess nachvollziehbar zu dokumentieren. Da es sich um kein klassisches systematic literature review (SLR) handelt, werden hinsichtlich des Vorgehens nur die Schritte durchgeführt, welche zur Identifizierung geeigneter Literatur beitragen. Dies beinhaltet die Suchstrategie, die Auswahlkriterien, die Qualitätsprüfung und eine gekürzte Form der Datenextraktion. Als Ergebis dieser Schritte soll somit eine Übersicht geschaffen werden, aus welcher hervorgeht, welche Arbeiten für welche Forschungsfragen relevant sein können. Der Stand der Forschung wird im Rahmen der Datensynthese in Abschnitt 2.5 zusammengetragen.

2.1 Suchstrategie

Für die Recherche werden hauptsächlich die Datenbanken ACM Digital Library (https://dl.acm.org/), IEEE Xplore (https://ieeexplore.ieee.org/) und Springer Link (https://link.springer.com/) verwendet.
Um die für die Suche benötigten Suchstrings zu erstellen, werden die Forschungsfragen thematisch sortiert und anschließend in ihre inhaltlichen Kernelemente zerlegt. Aus den resultierenden Listen mit Schlagworten können anschließend die Suchstrings kombiniert werden.
Die Forschungsfragen lassen sich in zwei Bereiche unterteilen. Zum einen gibt es den Bereich der Performance-Messung, welchen die Forschungsfragen RQ1 bis RQ4 betreffen und den Kern der Arbeit ausmachen.Die Forschungsfrage RQ5 fällt in den Bereich der Testdatengenerierung und ist als Vorarbeit für die Durchführung der Performance Messungen notwendig. Die Erfassung der Literatur erfolgt daher weniger umfangreich als für die Performance Messung und basiert auf einer stichwortartigen Suche für die folgende Schlagworte verwendet werden: generat*, synthetic, provenance, data, graph*, scalable, „DAG“, „provenance graph“.
Die Forschungsfrage RQ6 ist von der Recherche ausgenommen, da die Arbeit von Kocadag [45, S. 23ff] sich bereits mit der Identifikation von Speicherverfahren befasst hat. Einzig die Kriterien für die Auswahl des Speicherverfahrens ändern sich durch die unterschiedlichen Anforderungen. Daher werden die Ergebnisse der Literaturrecherche aus [45] als Grundlage in dieser Arbeit verwendet.

Suchbegriffe

Wissenschaftliche Arbeiten werden meist in englischer Sprache verfasst. Um die Suche darauf auszurichten, werden die Suchstrings ebenfalls mit englischen Begriffen formuliert. Zum Themengebiet Performance-Messung (RQ1-RQ4) ergeben sich die folgenden Suchbegriffe: *blockchain, ethereum, provenance graph, provenance data, measure, evaluate, metric, configuration, gas limit, PoS, PoA, performance, bloxberg.*

Suchstring für Performance-Messung

Um einen Überblick über die geeignete Literatur zu erhalten, wird ein initialer Suchstring aus allgemeinen Begriffen zur Performance-Messung erstellt. Aus der zuvor erstellten Liste werden hierzu die Begriffe *measure, evaluate, performance* und *ethereum* gewählt. Um die Begriffe zu verbinden, wird der boolsche Operator AND genutzt. Da für einige Begriffe unterschiedliche Varianten denkbar sind, werden die entsprechenden Segmente im Wort mit dem Platzhalter * (Asterisk) ersetzt. So repräsentiert bspw. *evaluat** sowohl *evaluate* als auch *evaluation.* Der resultierende Suchstring ist in Tabelle 2.1 mitsamt der gefundenen Ergebnisse pro Datenbank zu sehen. Im Anschluss wird der Suchstring basierend auf den Ergebnissen iterativ verfeinert, um die Ergebnisse weiter einzuschränken und die Qualität dieser zu verbessern.
Dafür wird der Suchsting anhand der *Index Terms* der gefundenen und passenden Literatur optimiert. So stellt sich nach wenigen Versuchen heraus, dass die Kombination aus *„performance evaluation“* und *ethereum* ein wünschenswertes Suchergebnis liefert. Zusätzliche Suchbegriffe, werden zunächst nicht in den Suchstring integriert, da die Anzahl der gefundenen Arbeiten bereits ein auswertbares Maß übersteigt. Stattdessen wird die Suche in Iteration vier experimentell eingeschränkt, anschließend aber wieder erweitert, da die Auswirkungen unzureichend abgeschätzt werden können. In Iteration fünf ist schließlich der finale Suchstring zu sehen, in welchem nur Themen im Bereich „machine learning“ ausgeschlossen werden. Außerdem wird die Suche um einige optionale Begriffe aus der zuvor erstellten Schlagwortliste erweitert. Die Anzahl der Ergebnisse ist anschließend zwar weiterhin zu hoch, kann aber über die Auswahlkriterien im nächsten Abschnitt ausreichend reduziert werden.

2.2 Auswahlkriterien

Nach den folgenden Kriterien wird die Literatur bewertet und entsprechende Quellen ausgewählt bzw. verworfen:

- K1: Die Literatur muss einen Bezug zu den Forschungsfragen (RQ1-RQ6) dieser Arbeit haben.
- K2: Die Literatur muss in englischer oder deutscher Sprache verfasst sein.
- K3: Der Titel der Literatur muss Performance, Ethereum, Bloxberg oder scalabil* enthalten.
- K4: Die Literatur muss im Bereich Computer Science angesiedelt sein (Springer Link)

Je nach Datenbank muss der Suchsting für die Kriterien K3 und K4 ergänzt werden. Für IEEE wird der folgende Zusatz angefügt: *AND (“Document Title“:performance OR “Document Title“:ethereum OR “Document Title“:scalabil* OR “Document Title“:bloxberg).* Für ACM wird

Iter.	Suchstring	ACM	IEEE	Springer	Überlegung
1	measur* AND evaluat* AND performance AND ethereum	1184	36	4504	Index Terms verwenden
2	“performance evaluation“ AND ethereum AND metric*	217	8	1622	Suche für IEEE erweitern
3	“performance evaluation“ AND ethereum	344	101	2059	Viele Treffer zu ML
4	“performance evaluation“ AND ethereum NOT (trading OR “machine learning“ OR artificial OR trade OR “cloud computing“)	79	70	98	Viele Einschränkungen, deren Berechtigung schwierig zu prüfen ist
5	“performance evaluation“ AND (ethereum OR blockchain) AND (scalabil* OR metric* OR provenance OR bloxberg OR “systematic survey“ OR slr) NOT “machine learning“	268	148	833	Ergebnisse weiter über den Titel selektieren
	Nach Anwendung der Auswahlkriterien (K1-K4)	26	45	48	

Tabelle 2.1: Suchstrings: Performance-Messung

die Suche über die „Advanced Search“ ausgeführt. Während in der Sektion „Search Within“ der normale Suchstring für „Anywhere“ eingetragen wird, kann über den Plus-Button ein weiteres Feld hinzugefügt werden. Dort wird „Title“ ausgewählt und der folgende Wert eingetragen: *performance OR ethereum OR scalabil* OR bloxberg*. Bei Springer Link wird die Einschränkung über die URL vorgenommen und drei einzelne Suchen für die jeweiligen Titel durchgeführt. Zu Beginn jeder Suche, wird für das Kriterium K4 der folgende Zusatz an die URL angehangen: *&facet-discipline=“Computer+Science“*. Anschließend folgt die Ausrichtung auf den Titel mit jeweils: *&dc.title=“performance“*, *&dc.title=“ethereum“*, *&dc.title=“scalabil*“* und *&dc.title=“bloxberg“*.

2.3 Qualitätsprüfung der Quellen

Um eine gute Qualität der verwendeten Quellen sicherzustellen, wird ein Formular mit Fragen definiert, welches für jede Quelle ausgefüllt wird. Die Fragen entsprechen Qualitätskriterien, welche sicherstellen sollen, dass die Literatur einen Beitrag zu den Forschungsfragen liefert und dem Aufbau einer wissenschaftlichen Arbeit entspricht. Abschließend werden zur Übersicht die mit „Ja“ beantworteten Fragen für die Bereiche „Allgemein zum Paper (Q)“ und „Bezug zu den Forschungsfragen (F)“ für jede Arbeit summiert. Der daraus resultierende Q- und F-Score soll eine Orientierung liefern, um die Quellen später anhand ihrer Qualität und Relevanz für die Arbeit auswählen zu können. Eine Ausnahme bildet das Qualitätskriterium F1, da dieses zur Datenextraktion dient und als Wert die behandelten Forschungsfragen enthält. Wenn das Feld nicht leer ist, erhöht es den F-Score dennoch um eins.

Allgemein zum Paper (Q):

- Q1: Sind die Ziele der Arbeit klar definiert?
- Q2: Sind Forschungsfragen definiert und werden diese Forschungsfragen beantwortet?
- Q3: Wird die Methodik der Arbeit nachvollziehbar beschrieben?
- Q4: Wird ein abschließendes Fazit gezogen?
- Q5: Werden die Ergebnisse der Arbeit kritisch reflektiert?

Bezug zu den Forschungsfragen (F):

- F1: Sind die Forschungsfragen für diese Arbeit relevant?
- F2: Werden Metriken zur Performance-Messung definiert?
- F3: Werden die Konfigurationen der Testnetze beschrieben?
  - F3.1: Wird der Einfluss unterschiedlicher Konfigurationen untersucht?
  - F3.2: Werden die Konsensalgorithmen PoS oder PoA untersucht?
- F4: Wird die Performanz im Bezug auf Provenance-Daten erfasst?
- F5: Werden die verwendeten Testdaten beschrieben oder zur Verfügung gestellt?
- F6: Wird die Skalierung der Performanz im Bezug auf die Datenmenge oder Anzahl der Knoten untersucht?
- F7: Wird ein Bezug zum Ethereum-Mainnet oder Bloxberg hergestellt?

Vor der Auswertung werden die Ergebnisse aus den Literaturdatenbanken exportiert. IEEE und Springer Link bieten hierzu den Export der Ergebnislisten als CSV-Datei an. Bei ACM wird als Format „ACM Ref“ gewählt, da dieses einfach ins CSV-Format überführt werden kann. Dazu werden die Punkte innerhalb der Datei, welche zur Trennung verwendet werden, durch Semikolons ersetzt und die Dateiendung in „csv“ geändert. Bei Springer Link sind die Felder durch Kommata getrennt und in Anführungszeichen gesetzt. Daher werden auch hier die Kommata durch Semikolons ersetzt, die Anführungszeichen entfernt und die Dateiendung in „csv“ geändert. Die CSV-Dateien werden abschließend geöffnet und im XLSX-Format gespeichert, um die Formatierung der Zellen zu ermöglichen. Anschließend werden die Arbeiten im PDF-Format aus den Literaturdatenbanken heruntergeladen.
Vor der eigentlichen Qualitätsprüfung, werden die Ergebnislisten vorsortiert. Hierzu werden der Titel, das Abstract und die Conclusion der Arbeiten oberflächlich geprüft. Erscheint eine Arbeit interessant und geeignet, wird sie in der entsprechenden XLSX-Datei grün markiert. Sind die Inhalte weniger passend, aber enthält z.B. gängige Metriken, wird die Arbeit gelb markiert. Die restlichen Arbeiten, welche generell zu wenige Berührungspunkte mit den voraussichtlichen Inhalten dieser Arbeit haben, werden rot markiert und aussortiert. Ebenso wie redundante Arbeiten, welche in mehreren Literaturdatenbanken enthalten sind. Durch diese Vorauswahl können die Ergebnisse von 119 auf 30 Arbeiten eingegrenzt werden. Die eigentliche Qualitätsprüfung entsprechend des definierten Formulars, wird anschließend in einer weiteren Exceltabelle durchgeführt.
Während des Ausfüllens der Qualitätskritieren, werden die Arbeiten nochmals genauer betrachtet und irrtümlich selektierte Quellen aussortiert. Nach Abschluss der Qualitätsprüfung verbleiben 17 Arbeiten für die Datenextraktion.

2.4 Datenextraktion

Die Extraktion der Inhalte erfolgt weniger dediziert, als von [44] beschrieben. Ziel der Extraktion ist es in diesem Fall einen Überblick über die Inhalte der Literatur zu gewinnen und so bei Bedarf schnell die richtigen Arbeiten finden zu können. Dafür wird in der Exceltabelle, welche auch die Qualitätsprüfung enthält, eine weitere Zeile angehangen. In dieser werden Schlagworte notiert, welche die Inhalte einer Arbeit zusammenfassen. Außerdem wird das Qualitätskriterium F1, anders als die anderen Fragen, nicht mit „Ja“ oder „Nein“ beantwortet. Stattdessen werden in dem Feld die behandelten Forschungsfragen notiert.

2.5 Datensynthese

Das Ergebnis der Literaturrecherche umfasst 17 Arbeiten, deren Vorgehen und Ergebnisse als wissenschaftliche Grundlage zur Beantwortung der Forschungsfragen dienen. Neben diesen wird weitere Literatur über Snowball Sampling bezogen und ebenfalls in diesem Abschnitt aufgeführt.
Aus den selektierten Quellen sticht insbesondere die Arbeit [34] von 2020 heraus, welche mittels einer systematischen Recherche unter anderem die Metriken, Techniken und Workloads zur Performance Evaluation untersucht. Dabei wird zwischen empirischen und analytischen Evaluationsmethoden unterschieden. Als Stärke der analytischen Modelle, wird vor allem die Analysemöglichkeit des Konsenslayers einer Blockchain-Plattform genannt. Weiter werden die betrachteten Modelle in vier Typen unterteilt und mit ihren Eigenschaften erfasst [34, S. 13]. Die empirischen Methoden werden ebenfalls in vier Kategorien unterteilt. In diese Kategorien fällt auch die identifizierte Literatur im Rahmen dieser Arbeit.
Die erste Kategorie bilden die *Benchmarks*, welche mit Hilfe von Frameworks für unterschiedliche Blockchain Systeme durchgeführt werden können. Die festgelegten Metriken und oft vordefinierten Workloads ermöglichen es besonders private Blockchain Systeme miteinander vergleichbar zu machen. Grund dafür ist das hohe Maß an Kontrolle über private Umgebungen, da Performance Tests idealerweise unter möglichst standardisierten Bedingungen durchgeführt werden sollten (vgl. [34, S. 6]). In dieser Kategorie betrachten die Autoren von [34] die Frameworks Blockbench [29], DAGbench [30] und Hyperledger Caliper [7]. Insbesondere Blockbench und Hyperledger Caliper eignen sich für Benchmarks mit Ethereum und betrachten ähnliche Metriken. Den größten Unterschied bilden die ausführbaren Workloads, welche bei Blockbench vordefiniert sind und bei Hyperledger Caliper selbst definiert werden müssen. Ergänzend fallen in diese Kategorie die Frameworks Gromit [56] und xBCBench [65] aus der Literaturrecherche. Gromit kann für Benchmarks auf jedem Blockchain System angewandt werden und fokusiert sich auf Metriken bezüglich der Transaktionen. Der Workload wird somit aus Wert-Transaktionen zwischen Konten erzeugt. xBCBench betrachtet wie Gromit ebenfalls den Durchsatz und die Latenz, aber erfasst zusätzlich einige Systemressourcen. Die Workloads besteht aus vorgefertigten Use Cases, welche für Ethereum bspw. das Anlegen von Accounts, Abfragen und Transaktionen zwischen Accounts beinhalten.
Die zweite Kategorie stellt das *Monitoring* dar. Die Autoren von [34] kommen zu dem Schluss, dass sich diese Methode am besten für die Analyse von öffentlichen Blockchain Systemen eignet, da dort einige Faktoren nicht beeinflusst werden können [34, S. 6]. Die Arbeit [69] präsentiert daher einen Log-basierten Ansatz, um das Verhalten des Netzwerkes während des Betriebs aufzuzeichnen. Dabei werden Abfragen auf dem Blockchain System selbst vermieden, um dieses möglichst nicht zu belasten. Die benötigten Daten werden stattdessen aus den

Log-Dateien der Teilnehmer bezogen und mittels Metriken ausgewertet. Einen anderen Ansatz stellt [57] vor, mit der Idee ein privates Netzwerk unter möglichst realistischen Bedingungen zu betreiben. Dazu wird die Wahrscheinlichkeitsverteilung der Zeitspannen zwischen den Transaktionen auf einem öffentlichen Zielsystem betrachtet. Anschließend werden darauf basierend Transaktionen auf einer privaten Blockchain ausgeführt und die zwei Frameworks Parity und Multichain evaluiert [57, S. 5].
Die dritte Kategorie beinhaltet die *experimentellen Analysen*, welche den Großteil der recherchierten Literatur ausmachen. Diese können gut an Testziele angepasst und unter kontrollierten Bedingungen durchgeführt werden.

Ethereum:
Schäffer et al. [60] betrachten den Einfluss verschiedener Parameter in privaten Ethereum Netzen und stellen fest, dass diese große Abhängigkeiten untereinander haben. Außerdem wird eine Hierarchie der Bottlenecks beschrieben, welche sich ebenso wie die Parameter gegenseitig beeinflussen.
An [60] anschließend, analysieren Leal et al. [48] ebenfalls den Einfluss der Konfigurationsparameter auf private Ethereum Netze. Hauptsächlich werden dafür die Konsensalgorithmen PoW und PoA untersucht, sowie feste und dynamische Gas Limits. Ziel der Arbeit ist es die ideale Konfiguration für eine zu sendenden Datenmenge zu finden. Als Methode wählen die Autoren hierzu Regressionsmodelle und halten abschließend ebenfalls, den starken Einfluss der Netzkonfiguration auf die Performance fest.

Hyperledger Fabric:
Die Arbeit [46] führt Benchmarks für verschiedene Workloads durch, um den Einfluss auf die Performance zu untersuchen. Dazu wird das Tool Caliper verwendet und der Durchsatz, die Latenz und die Skalierbarkeit betrachtet. Um die Werte zu ermitteln, werden Performance-Tests mit variierenden Workloads durchgeführt. Abschließend kommen die Autoren zu dem Schluss, dass die Resultate von der Konfiguration der Hardware, des Netzes und den verwendeten Smart Contract abhängen.
Dreyer et al. [31] untersuchen die Performance von Hyperledger Fabric v2.0 anhand des Durchsatzes, der Latenz und der Fehlerrate. Dabei wird eine Verbesserung in allen Bereichen im Vergleich zur vorherigen Hyperledger Fabric Version festgestellt. Außerdem wird der Einfluss anderer Komponenten (Peers, Orderers und Organizations) innerhalb des Netzes untersucht, mit dem Ergebnis, dass die Konfigurationen dieser einen großen Einfluss auf die Performance haben.
Die Arbeit [35] fokussiert sich auf den Hyperledger Besu Client und die Auswirkungen auf die Netzgröße, Senderate der Transaktionen oder das Load Balancing. Die Ergebnisse des Benchmarks mit Caliper werden vor allem durch die Blockzeit und die Blockgröße beeinflusst. Als weiteres Ergebnis stellt sich die Limitierung der Performance durch die Block Erzeugung und die Ausführung der Transaktionen heraus.

Andere Blockchain Systeme:
Basierend auf Multichain untersucht die Arbeit [49] die Auswirkungen unterschiedlicher Blockgrößen im Kontext von IoT Daten. Im Zuge dessen stellen die Autoren einige Probleme im Zusammenhang mit erhöhten Blockgrößen fest. So beschreiben sie eine Gefährdung der Dezentralität, der Sicherheit und eine Einschränkung der Skalierbarkeit.

Vergleich mehrerer Systeme:
Wang et al. [64] analysieren die Architekturen von Ethereum, Hyperledger Fabric, Hyperledger Sawtooth und Fisco-Bcos und führen anschließend Benchmarks mit dem Tool Caliper durch. In diesen werden insbesondere der Durchsatz und die Latenz als Metriken berücksichtig. Aus den Ergebnissen leiten die Autoren ab, dass die Performance gemeinschaftlicher Blockchains deutlich besser ausfällt, als die öffentlicher Blockchain Systeme.
Die Arbeit von Dabbagh et al. [26] betrachtet spezifischer die Performance von Hyperledger Fabric und Ethereum und verwendet dazu ebenso das Benchmarking-Tool Caliper. Mit diesem werden die Erfolgsrate, der Durchsatz, die durchschnittliche Latenz und der Ressourcenbedarf erfasst. Als Ergebnis des Benchmarks stellt sich Hyperledger Fabric hinsichtlich der durchschnittlichen Latenz, des Durchsatzes und der Ressourcen als performanter heraus. Ethereum dafür weist eine höhere Erfolgrate der Transaktionen auf.
Monrat et al. [55] vergleichen die PoA-Plattformen Ethereum, Corda, Quorum und Hyperledger Fabric im Kontext variierender Workloads und steigender Netzwerkgrößen. Als Metriken werden der Druchsatz und die Latenz gewählt, welche mit den Benchmark-Tools Caliper, Blockbench und Corda enterprise test suite ermittelt werden [10]. In den Resultaten erzielt Hyperledger Fabric die besten Werte.
Zu einem ähnlichen Ergebnis kommen die Autoren von [15], wobei diese Ethereum unter der Verwendung des PoW-Algorithmus, im Vergleich zu Hyperledger Fabric betrachten. Als Metiken wird die Zeit zum Erzeugen eines Blocks, die Blockgröße und die Latenz bei Transaktionen gewählt.
Auch die Arbeit [63] vergleicht die Performance von Ethereum und Hyperledger Fabric. Für Ethereum wird hierzu der Client Besu [6] und der PoA-Algorithmus verwendet. Hinsichtlich der Metriken wird der Fokus auf die Latenz und den Durchsatz gelegt und mittels Caliper ermittelt. In den Tests erreicht Hyperledger Fabric im Bereich des Durchsatzes und der Latenz bessere Ergebnisse. Die Werte brechen jedoch mit der Steigerung der Transaktionen schneller ein als bei Ethereum.

Die letzte Kategorie umfasst *Simulationen*. Ismail et al. [42] fassen die Funktionsweise unterschiedlicher Konsensalgorithmen, sowie einige Eigenschaften von Blockchain Plattformen für ihre Anwendungsmöglichkeiten zusammen. Außerdem werden der Einfluss der Blockgröße und der Anzahl der Knoten anhand des Bitcoin-Simulators untersucht. Als Metriken dienen die Bandbreite und die Ausführungszeit. Als Ergebnis halten die Autoren fest, dass die Performance im Bezug auf die Bandbreite mit der steigenden Anzahl an Validationen abnimmt, während eine größere Anzahl an Blöcke die Ausführungszeit verlängert. Außerdem könne die Performance deutlich verbessert werden, indem die Blockgröße erhöht wird.
Einige weitere Auswertungen zur allgemeinen Performance Evaluation von Blockchain Systemen sind in der Erhebung [34] aufgeführt.

Bei der Einordnung der Arbeiten fällt auf, dass sich die meisten im Rahmen von RQ4 befinden und somit Methoden oder Kriterien zum Messen der Performance behandeln. Der zweite Schwerpunkt liegt auf der Konfigurationen der Netze im Rahmen von RQ2. Die anderen Forschungsfragen (RQ1 und RQ3) werden nicht thematisiert. Der Bezug zum Speichern von Provenance-Graphen beim Erfassen der Performanz scheint somit weitgehend noch nicht betrachtet worden zu sein.
Häufig wird neben Ethereum die Plattform Hyperledger Fabric analysiert. Dies spiegelt sich in gewisser Weise auch in den betrachteten Benchmarks wieder, da für diese häufig das Tool Hyperledger Caliper genutzt wird. Mit dessen Hilfe werden im Rahmen von experimentellen

Analysen insbesondere der Durchsatz und die Latenz erfasst und evaluiert. In beiden Bereichen erzielt Hyperledger Fabric im Rahmen der Auswertung bessere Ergebnisse und ist somit meist performanter als Ethereum.

2.6 Abgrenzung zu anderen Arbeiten

Im Folgenden werden explizit Arbeiten abgegrenzt, welche Gemeinsamkeiten mit der vorliegenden Arbeit aufweisen. Ziel ist es die Unterschiede und den Fokus der Abschlussarbeit hervorzuheben.

Als eine der Grundlagen dient eine vorangegangene Abschlussarbeit [45], welche eine systematische Literaturrecherche zum Speichern von Provenance Daten in einer Blockchain, sowie die Implementierung eines solchen Ansatzes durchgeführt hat. Dafür wurde ein Verfahren gewählt, welches die Hashwerte der Provenance Daten in einer Blockchain speichert und die eigentlichen Daten außerhalb persistiert. Viele Arbeiten behandeln Hyperledger Fabric Einiges im IoT Bereich Oft die gleichen Metriken verwendet Eine Arbeit untersucht eine spezielle Form von Provenance Daten [35] untersucht Besu im Hinblick auf Skalierung (horizontal/vertikal) und Block size mit Caliper. [49] erhöht die Blocksize für Anwendungsfälle im IoT Bereich. Nutzt Multichain und PoA. Sendet aber viele kleinere Transaktionen und beobachtet die Eignung unterschiedlicher Block sizes. Die Arbeit [60] führt Performance-Tests für PoW und PoA Netze mit Geth durch. Eine der Forschungsfragen richtet sich ebenfalls nach geeigneten Metriken für die Durchführung der Messungen. Neben den oft verwendeten Metriken Latenz und Durchsatz, wird hier zusätzlich die Skalierbarkeit aufgeführt. Einen weiteren Schwerpunkt macht die Konfiguration der Testnetze aus. Für das PoA-Netz werden die block period und das gas limit verändert, um den Effekt auf den Durchsatz und die Latenz zu untersuchen. Der Workload wird über zwei Smart Contract generiert. Der erste führt Wertgutschriften zwischen zwei Adressen durch, während der andere seinen eigenen Status abfragt. Hinsichtlich der Skalierbarkeit, werden die CPU Kerne, der Arbeitsspeicher und die Anzahl der Knoten für die Netze variiert.
Die Hauptunterschied zwischen der Arbeit von [60] und der vorliegenden Arbeit, finden sich im verwendeten Workload. So wird die Performance der Netze nicht anhand größerer zu speichernder Daten betrachtet. Weiterhin werden die Messungen nicht für PoS Netze oder den Besu-Client durchgeführt.

Das in der Arbeit [61] entwickelte Verfahren zum Speichern von Provenance-Daten, wird in leicht abgewandelter Form für diese Arbeit übernommen. In der ursprünglichen Arbeit wird ein für den IoT Bereich angepasstes Provenance-Datenmodell angewendet. Für das Speicherverfahren wird der Gas-Bedarf in Abhängigkeit zur Anzahl der zu speichernden Zeichen gemessen [61, S. 10]. Außerdem wird der Gas-Bedarf mit steigender Anzahl der Provenance-Einträge ermittelt. Im zweiten Schritt wird die Latenz für die beiden öffentlichen Testnetze Rinkeby und Ropsten erfasst und der Einfluss der Priorisierung durch den Gas Preis betrachtet. Für die gleichen Testnetze wird außerdem der Durchsatz, ebenfalls mit unterschiedlichen Gas Preisen gemessen.
Aspekte der Skalierung hinsichtlich der Anzahl der Knoten oder die Anwendung für große Provenance-Daten, werden nicht betrachtet.

Die Autoren von [35] betrachten den Besu-Client im Rahmen privater Blockchain Netze. Zur Vereinfachung setzen diese das gas limit und das Limit für die Größe des Contracts

auf das Maximum, während der minimale Gas-Preis und die Schwierigkeit auf das Minimum gesetzt werden. Durchgeführt werden die Performance-Tests mit dem dem Benchmark-Tool Hyplerledger Caliper. Zur Generierung des Workloads werden die vordefinierten Transaktionstypen von Caliper genutzt. Als Metriken werden der Durchsatz, die Latenz, der Ressourcenbedarf der Docker Container und die Skalierbarkeit gewählt. Letztere wird sowohl vertikal, als auch horizontal untersucht.

Kapitel 3

Grundlagen

3.1 Data Provenance

3.1.1 Definition

Amarnath Gupta definiert den Begriff *Data Provenance* als „[...] record trail that accounts for the origin of a piece of data (in a database, document or repository) together with an explanation of how and why it got to the present place.“ [39, S. 812].
Das World Wide Web Consortium (W3C) definiert den Begriff der *Provenance* im Zusammenhang mit den PROV Dokumenten als „[...] information about entities, activities, and people involved in producing a piece of data or thing, which can be used to form assessments about its quality, reliability or trustworthiness.“ [38].

3.2 Ethereum

Ethereum ist eine dezentrale, Open-Source-Blockchain-Plattform, die im Juli 2015 von Vitalik Buterin und anderen Entwicklern eingeführt wurde. Im Gegensatz zu Bitcoin, das primär als digitales Zahlungsmittel dient, wurde auch Ethereum entwickelt, um u.a. Smart Contracts (vgl. "3.2.3 Smart Contract") zu unterstützen. [17]
In einer Blockchain sind mehrere Knoten in einem Netz verteilt. Blockchains arbeiten mit Konsensverfahren, um Informationen aufzunehmen und an andere Knoten zu verteilen. Die Architektur von Ethereum nach der Enterprise Ethereum Alliance (EEA) [67, S. 211]. Das Konsensverfahren stell sicher, dass der Inhalt der Blockchain trotz dezentraler Verarbeitung konsistent bleibt.

3.2.1 Proof of Stake

Ethereum verwendet seit 2022 als Konsensmechanismus den sogenannten Proof of Stake (PoS). [18]
Der Proof of Stake (PoS) ist ein Konsensmechanismus, bei dem bestimmte Teilnehmer für die Korrektheit ihrer Prüfung eines Blocks eine finanzielle Sicherheit hinterlegen (sogenanntes Staken). Die validierenden Teilnehmer, auch Validatoren genannt, erhalten für ihre Prüfung eine Belohnung oder verlieren ihre hinterlegte Sicherheit, wenn sie betrügen oder fehlerhaft arbeiten. [67, S. 213] [19]

Im Gegensatz zum Proof of Work (PoW) ist ein PoS u.a. energieeffizienter, da dieser Konsensmechanismus nicht auf der kompetitiven Lösung eines Rechenproblems basiert. Zudem wird

eine zu große Konzentration von Rechenleistung, wie sie beispielsweise in einzelnen Mining Pools bei PoW vorkommt, bei PoS vermieden. Dies fördert die Dezentralisierung des Netzwerk. PoS Mechanismen gelten gemeinhin als komplexer in der Implementierung als PoW Mechanismen. [19]

3.2.2 Proof of Authority

Das Konsensverfahren Proof of Authority (PoA) basiert darauf, dass bestimmten Teilnehmern vertraut wird und diese mit der Reputation, die mit ihrer Identität verbunden ist, bei der Erstellung neuer Blöcke für ihre Korrektheit bürgen. Vertrauenswürdige Teilnehmer können durch verschiedene Prozesse, beispielsweise durch Wahlen bereits ernannter vertrauenswürdiger Teilnehmer, hinzugefügt oder entfernt werden. [17][67, S. 213]

PoA Ansätze eignen sich vor allem in Fällen, in denen vertrauenswürde Teilnehmer gut benannt werden können und werden deshalb oft in private Chains oder Testnets eingesetzt.
PoA hat im Gegensatz zu PoS und PoW durch den verkleinerten Kreis an Validatoren typischerweise eine höhere Geschwindigkeit. Die Sicherheit der Blockchain hängt von der konkreten Ausgestaltung der Auswahl der vertrauenswürdigen Teilnehmer ab. [20]

3.2.3 Smart Contract

Ein Smart Contract ist ein Programm, das auf einer dezentralen Blockchain gespeichert und ausgeführt wird. Es können durch verschiedene Interaktionen in einem Smart Contract definierte Aktionen ausgelöst werden. Ein Smart Contract wird üblicherweise in einer Programmiersprache wie Solidity geschrieben und dann in EVM (Ethereum Virtual Machine)-Bytecode übersetzt. In der EVM-Laufzeitumgebung führen Nodes im Netzwerk den Smart Contract aus. Das Ergebnis einer Ausführung wird gemeinsam mit dem Zustand des Smart Contracts in einem Block gesichert. Das Ergebnis muss unter den ausführenden Nodes einen Konsens haben. [21][29, S. 3]
Die Ausführung von Aktionen in einem Smart Contract erfolgt durch das Senden von Transaktionen an die Adresse des Smart Contracts. Diese Transaktionen enthalten die notwendigen Daten und Befehle, um den Contract zu aktivieren und die im Code festgelegten Aktionen auszuführen. [22]
Sowohl die Erstellung als auch die Ausführung von Smart Contracts kosten Gebühren. Diese Gebühren decken die Rechenleistung und Speicherressourcen, die zur Ausführung des Contracts erforderlich sind.
Ein Smart Contract kann nur auf Daten innerhalb der Blockchain zugreifen. Externe Informationen müssen über spezielle Dienste (sogenannte Oracles) in der Blockchain hinterlegt werden und somit Smart Contracts zugänglich gemacht werden.
Typische Anwendungsfälle für Smart Contracts sind beispielsweise dezentrale Anwendungen (DApps) und der Tausch von Kryptowährungen. DApps laufen ohne zentrale Kontrolle und nutzen Smart Contracts, um die Geschäftslogik und Transaktionen zu automatisieren. [23]

3.2.4 Ethereum Knoten

Ein Ethereum Knoten besteht aus einem *Execution Client* und einem *Consensus Client* und kann sich mit anderen Ethereum Knoten zu einem Netzwerk verbinden.
Der Execution Client erfasst neue Transaktionen im Netzwerk, verteilt diese und führt sie in der EVM (Ethereum virtual machine) aus. Außerdem speichert er den neusten Stand aller

Daten im Ethereum Netz [24].

Der Consensus Client führt eine Implementierung des Proof-of-Stake Konsensalgorithmus aus, über welchen innerhalb des Netzes entschieden wird, welche Blöcke an die Blockchain angefügt werden [24].

3.2.5 Architektur

Die Enterprise Ethereum Alliance (EEA) hat den Architektur Stack von Ethereum in einer Übersicht aufbereitet [14] und in fünf Schichten[67] unterteilt.

Application
Die oberste Schicht stellen die Anwendungen dar. Wie bei anderen Blockchains gibt es Tokens, welche eine Währung innerhalb des Blockchain Systems sind. Außerdem werden in dieser Schicht Smart Contracts, die Web3-Schnittstelle, sowie diverse Oberflächen für die Interaktion mit der Blockchain gruppiert.

Tooling
Ethereum stellt einige Tools und Schnittstellen für Entwickler und dezentrale Anwendungen zur Verfügung. Diese unterstützen beim Entwickeln und Testen von Smart Contracts. Für allgemeine Anwender werden unterschiedliche Wallets angeboten, um dezentrale Anwendungen zu nutzen. Die Kommunikation zwischen einem Knoten und externen Clients, findet meist über das RPC Protokoll statt.

Enterprise
Der Enterprise Layer enthält Elemente, welche nicht von der öffentlichen Ethereum Blockchain unterstützt werden, aber für Unternehmen von Interesse sind. Zunächst gibt es keine Privatsphäre, da Transaktionen öffentlich sind. Weiterhin gilt die Performance nicht als optimal. Abschließend gibt es keine Berechtigungen für öffentliche Blockchains, was zu Sicherheitsrisiken führen kann.

Core Blockchain
Die Kernfunktionalitäten der Ethereum Blockchain machen der Konsensalgorithmus, die EVM und der Speicher aus. In Ethereum wird der Konsensalgorithmus Proof-of-Stake verwendet. Die Ethereum Virtual Machine führt den Bytecode eines Smart Contracts aus und ermöglicht so die Umsetzung dezentraler Anwendungen auf Basis der Blockchain. Das Speichern von Daten in einer Blockchain ist verhältnismäßig teuer. Für Transaktionen fallen Gebühren an, welche mit der Beauftragung hinterlegt werden müssen.

Network
Client Nodes verbinden sich erst mit einem Bootnode und erhalten über diesen die enode Informationen anderer Peers. Anschließend verbinden sie sich mit den Peers. Die verbundenen client Nodes bilden die Ethereum Blockchain [67, S. 212].

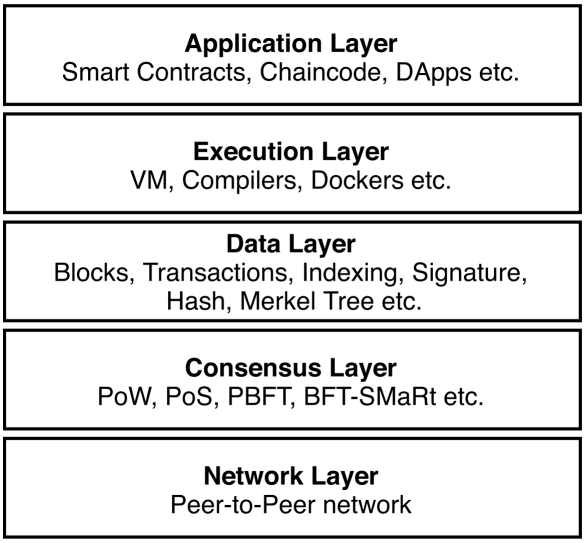

Abbildung 3.1: Abstraktionsschichten-Modell - entnommen aus [34, S. 4]

3.2.6 Abstraktionsschichten

[29, S. 4] hat ein allgemeines Schichtmodell für den Aufbau von Blockchain Systemen erstellt, mit dem Hintergrund den Workload auf einzelne Schichten auszurichten. Somit kann eine einzelne Schichten über das Test-Design gezielter ausgelastet und mögliche Bottlenecks eingegrenzt werden. Das Modell umfasst den Application, Execution, Data Model und Consensus Layer.
Eine weitere, jedoch nicht auf Performance Tests ausgelegte Architektur wird in [16, Layered structure of the blockchain architecture] beschrieben. [34, S. 4] kombiniert beide Modelle, um das ursprüngliche Modell von [29] zu verfeinern und unterteilt eine Blockchain im Ergebnis in fünf Schichten, welche in Abbildung 3.1 zu sehen sind.

Application Layer: - Nutzung durch Endnutzer - Kryptowährung - DApps Der Application Layer ist die oberste Schicht und präsentiert das Ergebnis einer Aktion. Sie wird von allen anderen Schichten beeinflusst und repräsentiert daher nach [34, S. 4] die Gesamtperformance. **Execution Layer:** Führt den Smart Contract bzw. dessen Bytecode innerhalbt eines Knotens aus, wenn die Blockchain dies unterstützt. Die Ausführung eines Smart Contracts muss deterministisch sein, da alle Knoten zu dem gleichen Resultat kommen müssen. Der Layer ist nach [34, S. 4] von den zur Verfügung stehenden Ressourcen für den Knoten abhängig. **Data Layer:** Beinhaltete alles was mit Datenstrukturen, Transaktionsmodellen und Verschlüsselung zutun hat. Hier beeinflussen die Hash-Funktionen, Blockgröße und verwendeten Datenstrukturen die Performance.
Consensus Layer: Wird eine neue Transaktion eingereicht, definiert der Konsensalgorithmus die Regeln nach welchen die Transaktion genehmigt und verteilt wird. Die Konsensalgorithmen lassen sich in proof-based und vote-based Algorithmen unterteilen. Die vote-based Algorithmen sind performanter, beruhen auf einem Nachrichtenaustausch und finden vorrangig in permissioned Blockchains ihren Einsatz. Die proof-based Algorithmen werden häufig in öffentlichen Blockchains verwendet (PoW, PoS, GHOST, PoET). Außerdem gibt es auch Hybrid-Varianten, welche beide Typen miteinander kombinieren, um die Vorteile beider Algorithmen zu verbinden.

Network Layer: Die Knoten einer Blockchain sind über ein peer-to-peer (P2P) Netzwerk miteinander verbunden. Der Layer ist besonderes in Blockchains relevant, in denen viel Kommunikation notwendig ist. Die Synchronisation der Knoten und die Peer discovery sind ebenfalls vom Netzwerk abhängig. Daher haben typische Werte wie der Paketverlust und die Verzögerung durch das Netz einen Einfluss auf die Performanz.

3.3 Synthetische Daten

Die Nutzung synthetische Daten bietet viele Vorteile. So können sie genutzt werden, um Hypothesen und Modelle in einer kontrollierten Umgebung zu testen. Dafür können sie gezielt und in beliebiger Anzahl, zur Abbildung bestimmter Szenarien erzeugt werden und bieten daher mehr Kontrolle. Da die Daten für einen speziellen Zweck generiert werden, kann sogar die Qualität höher sein, als die echter Daten. Denn diese können durch ihre Größe, Vielfalt und unterschiedliche Quellen unhandlich werden. Außerdem kann das Problem auftreten, dass sie die Wirklichkeit zu verzerren und somit nicht repräsentativ zu sein. Auch dies kann mit generierten Daten vermieden werden.
Ein weiteres Anwendungsfeld ist die das Trainieren und Testen von Machine Learning Modellen. Diese Modelle können auch genutzt werden, um selbst Daten zu generieren. [40, S. 2f].
Ein anderer Aspekt kann der Schutz der Privatsphäre und sensibler Informationen sein. Das Journal *The Annual Review of Statistics and Its Application* beschäftigt sich in dem Artikel [59] mit synthetischen Daten, welche auf Grundlage erhobener Daten durch Befragte erzeugt werden. Mithilfe statistischer Verfahren werden Informationen aus den echten Datensätzen extrahiert und stattdessen synthetische Daten für den öffentlichen Gebrauch verwendet. Die Herausforderung der Verfahren besteht darin, die Identitäten der Befragten zu schützten und gleichzeitig den Nutzen (utility) der Daten zu erhalten [59, S. 131f].

Definition:
Synthetische Daten sind keine echten Daten, werden aber auf Basis dieser erzeugt und haben die gleichen statistischen Eigenschaften. Die Genauigkeit mit welcher ein synthetischer Datensatz einen echten Datensatz vertritt, ist ein Maß für die Nützlichkeit (utility). Die erzeugten Daten können dabei unterschiedlicher Art sein. Zum einen strukturierte Daten aus bspw. einer Datenbank und zum anderen unstrukturierte Daten wie Bilder, Videos oder Texte [32, Ch. 1: Defining Synthetic Data].

Eine weitere Definition wird in dem Buch *Synthetic Data for Deep Learning* [40, S. 2] hergeleitet und charakterisiert synthetische Daten durch die folgenden Attribute:

- Nicht durch direkte Messung ermittelt
- Von einem Algorithmus generiert
- Mit mathematischen oder statistischen Modellen verbunden
- Ahmen echte Daten nach

Diese Definition erweitert die von [32] vor allem um den Punkt, dass synthetische Daten von einem Algorithmus generiert werden.

Welche gültigen Schlussfolgerungen aus synthetischen Daten gezogen werden können, muss für jeden Prozess, der die Daten erstellt, sichergestellt werden.

3.4 Performance Tests

Performance Test ist ein Überbegriff für unterschiedliche Arten von Tests [66, S.] Performance Tests umfassen die Erstellung und Ausführung der Tests, ebenso wie die Interpretation der Ergebnisse, sowie die Untersuchung von Fehlern und Defekten. Dabei verfolgen sie messbare Ziele, anhand derer Erfolg und Misserfolg der Tests beurteilt werden kann (siehe auch [66, S. 70f]) Der Testprozess muss wohldefiniert und gleich bleiben. Stakeholder sollen mit Informationen versorgt werden, auf deren Basis sie weitere Entscheidungen treffen können.[66, S. xix] Die Betrachtung von Zeitfaktoren ist häufig interessant, da sie die Fähigkeit eines Systems zum Reagieren innerhalb eines Zeitintervalls unter bestimmten Umständen prüft [66, S. 9]. Sie liefern quantifizierbaren Metriken, um zu zeigen, ob die Performance Anforderungen erfüllt wurden oder inwiefern das System verbessert werden muss, um sie zu erfüllen [66, S. 34].
Man unterscheidet bei Tests zwischen statischen und dynamischen Tests.
Statische Tests betrachten die Architektur, und stellen statische Anforderungen an die Code Segmente. So können Defekte bereits in frühen Entwicklungsphasen entdeckt und beseitigt werden. Bei dynamischen Tests werden Einheiten des zu testenden Systems ausgeführt. [66, S. 40]

Kapitel 4

Konzepte

In diesem Kapitel wird die Konzeption der Prototypen, sowie die Planung der Performanzmessung beschrieben. Im Hinblick auf die Prototypen richtet sich der Fokus auf die Entwicklung der Testverfahren und dem Aufsetzen der Testumgebung. Dies hat den Hintergrund, dass es keine Nutzergruppen gibt und die Anforderungen aus den Forschungsfragen der Arbeit abgeleitet werden.

4.1 Generierung synthetischer Provenance Graphen

Die Erzeugung von Graphen wird in mehreren Arbeiten behandelt. Die Ziele des Ansatzes von [36] kommen denen der vorliegenden Arbeit am nächsten, da auch dort ein Provenance-Graph nach dem Modell von W3C [54] erzeugt werden soll. Die Generierung erfolgt dabei aber in wesentlichen Teilen mit Hilfe der Abragesprache Cypher und der Datenbank Neo4j [36, S. 25f]. Einen weiteren Ansatz liefert das Framework Flurry [43], welches die Provenance-Daten aus Vorgängen in einem Linux-System generiert. Dazu nutzt es die Bibliotheken CamFlow und libprovenance, welche unterschiedliche Aspekte in Form von Provenance-Graphen aufzeichnet. Eine Variante zur Generierung von Provenance-Graphen liefert der Generator ProvGen [36]. Der Ansatz bietet die Möglichkeit diverse Parameter des Graphen zu konfigurieren. Als Ergebnis erzeugt der Generator einen Provenance-Graph entsprechend des W3C Standards [54]. Dementsprechend wird in dem Ansatz auch die Kompatiblität der unterschiedlichen Kantentypen im Bezug auf die verschiedenen Knotentypen behandelt. Zusätzlich haben die Autoren den Anspruch die Generierung des Graphen über einen Seed und anpassbare Bedingungen beeinflussen zu können. Die Implementierung funktioniert jedoch nur im Zusammenspiel mit der Datenbank Neo4j, da sie auf der Abfragensprachen Cypher beruht [36, S. 25f].
Einen anderen auf Python basierenden Ansatz hat die Arbeit [25] mit dem Generator XL-STaGe entwickelt. Dieser kann gerichtete azyklischen Graphen generieren und bietet ebenfalls Möglichkeiten den Generierungsprozess des Graphen zu beeinflussen. Neben der Anzahl der Knoten und der Tiefe des Graphen, kann die Form des Graphen und die Wahrscheinlichkeit für Kanten zwischen entfernteren Knoten angegeben werden [25, S. 3]. Ein weiteres Ziel der Arbeit ist es aus dem generierten Graph ein Verkehrsmodell für Prozesse zu erzeugen. Dieser Aspekt ist für die Generierung des Graphen jedoch unerheblich.

4.1.1 Anforderungen an den Datengenerator

Die Anforderungen werden gemäß der Empfehlungen von Chris Rupp und Klaus Pohl [58, S. 64] mittels Satzschablone formuliert. Diese soll die Strukturierung und Formulierung natürlichsprachlicher Anforderungen vereinfachen und Effekte wie Mehrdeutigkeit oder

passive Sätze reduzieren. Außerdem werden durch den Einsatz der Schablone mehrere Anforderungen in einem Satz vermieden.
Die verwendete Schablone ist in [58, S.72] zu finden und ermöglicht es, mithilfe definierter Satzbausteine eine Anforderung zu formulieren.

Funktionale Anforderungen

F1 Der Testdatengenerator wird vom Nutzer über den Quellcode konfiguriert.

F2 Der Testdatengenerator wird vom Nutzer über den Python Interpreter ausgeführt.

F3 Der Testdatengenerator kann über die folgenden Parameter konfiguriert werden:

- Wahrscheinlichkeiten für Relationen
- Name der Ausgabedateien (unterscheiden sich in der Dateiendung)
- Anzahl der Knoten
- Tiefe des Provenance-Graphen
- Max. Breite des Provenance-Graphen

F4 Der Testdatengenerator muss alle Relationen des W3C Modells [54] unterstützen: wasGeneratedBy, used, wasInformedBy, wasDerivedFrom, wasAttributedTo, wasAssociatedWith, actedOnBehalfOf, wasInfluencedBy, wasStartedBy, wasEndedBy, wasInvalidatedBy, alternateOf, specializationOf und hadMember.

F5 Der Testdatengenerator muss die folgenden Knotentypen des W3C Modells [54] unterstützen: Agent, Activity und Entity

F6 Der Testdatengenerator kann Provenance-Graphen gemäß der konfigurierten Parameter generieren.

F7 Der Testdatengenerator kann Provenance-Graphen bis zu einer Größe von 100 Knoten graphisch im PNG Format speichern.

F8 Der Testdatengenerator muss die generierten Provenance-Graphen gemäß der PROV-N Notation von W3C [53] speichern.

F9 Der Testdatengenerator muss Provenance-Graphen mit bis zu 50.000 Knoten erstellen können.

Qualitätsanforderungen

Q1 Der Testdatengenerator kann Provenance-Graphen mit bis zu 1000 Knoten in unter 5 Minuten erstellen und speichern.

Q2 Das Validator Tool [12] weist die erzeugte PROVN-Datei als valide aus.

Rahmenbedingungen

R1 Der Testdatengenerator muss auf einem Standard-Notebook des DLR ausführbar sein.

R2 Der Testdatengenerator muss in Python implementiert werden.

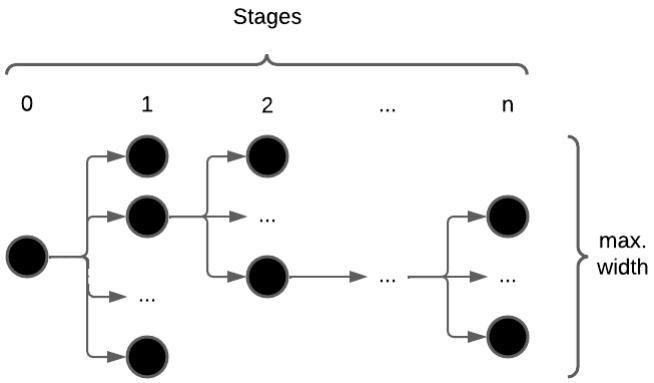

Abbildung 4.1: Testdatengenerator - Graph Konzept

4.1.2 Geplantes Vorgehen zur Erzeugung des Graphen

Das Vorgehen wird größtenteils unabhängig von den recherchierten Ansätzen geplant. Erst für die Visualisierung werden Teile des Generators XL-STaGe [25] verwendet. Um später einfacher an dessen Implementierung anschließen zu können, wird zuvor bereits der gedankliche Ansatz übernommen, die Knoten in *Stages* aufzuteilen [25, S. 3]. Zudem werden die Knoten und Kanten separat voneinander gespeichert.
Der Prozess der Generierung wird anhand von Abbildung 4.1 erläutert. Gemäß der beschriebenen Parameter in Anforderung *F3*, sollen die Dimensionen des Graphen beeinflusst werden können. Dazu zählt die Anzahl der Knoten, die maximale Breite und die Tiefe des Graphen. Die Tiefe wird in Anlehnung an die Arbeit [25] im Folgenden in Stages angegeben. Zuerst wird die Verteilung der Knoten innerhalb des Graphen zufällig festgelegt. Wichtig ist dabei, dass sich in Stage 0 nur ein Knoten befinden darf und sich die Verteilung nicht außerhalb der gesetzten Parameter erstreckt. Für die Generierung der Graphelemente werden anschließend alle Stages schrittweise durchgegangen. Stage 0 stellt bei dem Ablauf eine Ausnahme dar, weil zu dieser keine eingehenden Relationen erzeugt werden müssen. Für diese wird nur eine ein Startknoten eines zufälligen Knotentyps generiert. Ab der nächsten Stage dann die Relationen zwischen der jeweils aktuellen Stage *s* und der vorherigen Stage *s-1* gebildet. Stage *s-1* ist für die Bestimmung der Relationen wichtig, da diese nicht zwischen allen Knotentypen definiert sind. Bei der Erstellung einer Relation wird als erstes zufällig ein Startknoten aus der Stage *s-1* bestimmt. Anschließend wird eine mit diesem Knoten kompatible Relation ausgewählt. Die Auswahl wird außerdem auf Basis einer inital angelegten Wahrscheinlichkeitsverteilung für alle Relationen aus der Anforderung *F4* getroffen. Abschließend werden die Endknoten in Stage *s* angelegt, welche sich aus den angelegten Relationen ergeben. Für den Fall, dass mehrere Knotentypen für eine Relation möglich sind, wird dieser zufällig bestimmt.

4.1.3 Architektur des Datengenerators

Der Generator basiert auf einem objektorientierten Ansatz, dessen Struktur in Abbildung 4.2 zu sehen ist. Der Ablauf des Programms, wird von der Klasse *StructureGenerator* gesteuert. Der zu generierende Provenance-Graph basiert grundsätzlich auf Knoten und Kanten. Diese werden von den Klassen *Edge* und *Node* repräsentiert. Beide haben jeweils einen durch das PROV Data Model [54] definierten Typ, welcher über die Enums *EdgeType* und *NodeType* repräsentiert wird. Die Knoten können zusätzlich einen Prefix vor ihrem Bezeichner haben, welcher am Anfang eines Dokuments über einen Namespace angelegt werden kann. Einige

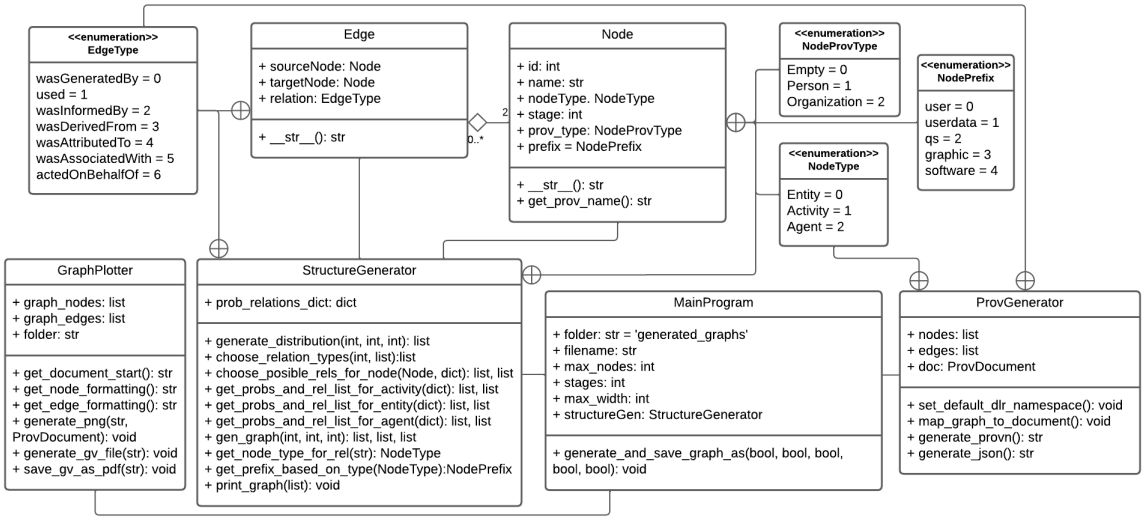

Abbildung 4.2: Testdatengenerator - UML Klassendiagramm

vom DLR definierte Namespaces werden über die Methode *set_ default_ dlr_ namespace* in der Klasse *ProvGenerator* gesetzt. Die dort definierten Prefixes stellt das Enum *NodePrefix* zur Verfügung. Eine weite Definition erfolgt über das Enum *NodeProvType*, in welchem exemplarisch drei Einträge enthalten sind. Diese können genutzt werden, um die Knoten mit zusätzlichen Informationen anzureichern. Die so generierbaren Knoten können beliebig vielen Kanten zugewiesen werden, während eine Kante immer zwei Knoten miteinander verbindet.
Die Klasse *StructureGenerator* nutzt die beiden Klassen, um den Graph zu generieren. Anschließend wird über die Klasse *ProvGenerator* das PROVN-Dokument erzeugt. Dafür wird das Python Package *prov* verwendet.
Neben der Generierung des PROVN-Dokumentes, ist es auch möglich den erzeugten Provenance-Graphen als PNG oder PDF zu exportieren. Bereitgestellt werden diese Funktionen von der Klasse *GraphPlotter*. Neben dem Exportieren als PNG über das *prov*-Package, implementiert die Klasse Funktionen, um den Provenance-Graphen als GV- und PDF-Datei zu speichern.

4.2 Auswahl der Speicherverfahren

In der Abschlussarbeit [45] wird eine systematische Literaturrecherche zum Speichern von Provenance-Graphen durchgeführt und die Eignung identifizierter Verfahren gegeneinander abgewogen [45, S. 23ff]. Von Interesse sind die On-Chain Verfahren, bei welchen der gesamte Graph innerhalb der Blockchain gespeichert wird. Dabei wird zwischen dem Speichern der Dokumente, der Elemente und einer kanonisierten Form unterschieden. Insgesamt werden 11 Arbeiten betrachtet. Zur Auswahl des Verfahrens werden folgende Kriterien festgelegt:

- K1 Das Verfahen kann einen gesamten Provenance-Graph speichern
- K2: Das Verfahen ist ausführlich genug für eine eigene Implementierung beschrieben
- K3: Es ist eine Implementierung vorhanden, auf die zugegriffen werden kann.

Arbeit	K1	K2	K3	K4
[27]	Ja	Ja	Nein	Nein
[61]	Ja	Ja	Ja	Ja
[62]	Ja	Ja	Nein	Nein
[28]	Ja	Ja	Nein	Nein
[68]	Ja	Nein	Nein	Nein
[41]	Ja	Ja	Nein	Nein
[47]	Ja	Nein	Nein	Nein
[52]	Ja	Ja	Ja	Nein
[51]	Ja	Nein	Nein	Nein
[33]	Nein	Nein	Nein	Nein
[50]	Nein	Ja	Ja	Nein

Tabelle 4.1: Auswahl des Speicherverfahrens

- K4: Das Verfahren ist für die Blockchain Ethereum hinterlegt und steht somit als Smart Contract in der Sprache Solidity zur Verfügung.

Die Ergebnisse der Auswertung sind in Tabelle 4.1 zu finden. Nach Auswertung der Verfahren ist keine weitere Gewichtung notwendig, da nur das Verfahren [61] alle vier Kriterien erfüllt und somit ohne größere Anpassungen für den Zweck der Arbeit verwendet werden kann.

4.3 Möglichkeiten zur Bestimmung der Performanz

In der Ausarbeitung [34] werden Performance Defizite beim Durchsatz und der Latenz beschrieben. Weiter wertet die Arbeit unterschiedliche Ansätze zum Evaulieren der Performance von Blockchains aus. Die Autoren nehmen dabei eine Unterscheidung zwischen empirischen und analytischen Methoden vor. Verfahren zu empirischen Evaluation werden in weitere Untergruppen unterteilt: Performance Benchmarking, Monitoring, experimentelle Analyse und Simulation.
Um die Ergebnisse mit denen anderer Arbeiten besser vergleichbar zu gestallten, sollen die Performance-Tests mit einem Benchmark-Tool durchgeführt werden. Nur zur Beantwortung der Forschungsfrage *RQ3* sollen experimentell Tests verwendet werden.

4.4 Metriken

Im Zuge der Forschungsfrage *RQ4* sollen Kriterien gefunden werden, um den Speichervorgang eines Provenance-Graphen messbar zu machen. Interessant ist dabei in erster Linie die Dauer des Speichervorgangs. Diese wird durch die Latenz einer Transaktion erfasst, welche die Zeit zwischen der initialen Beauftragung t_1 und der Erzeugung des Blocks t_2 meint [48, S. 6]. Formal kann die Latenz L wie folgt berechnet werden: $L = t_2 - t_1$.
Wenn mehrere Transaktionen durchgeführt werden, ist außerdem der Durchsatz von Interesse, welcher die Anzahl an Transaktionen beschreibt, die pro Sekunde in einem Block gespeichert werden [26, S. 3]. Die Anzahl der Transaktionen pro Sekunde wird auch mit *tps* abgekürzt.
Die Latenz und der Durchsatz werden in den meisten der recherchierten Arbeiten genutzt, um die Performance zu messen. Als Beispiele sind [60, S. 5], [35, S. 4] und [63, S. 5] zu nenne. Zudem weisen die Autoren der Arbeit [34] an einigen Stellen ihrer Erhebung diese beiden Metriken aus. Für die ausgewählten Metriken spricht zudem, dass die Autoren der Arbeiten [60,

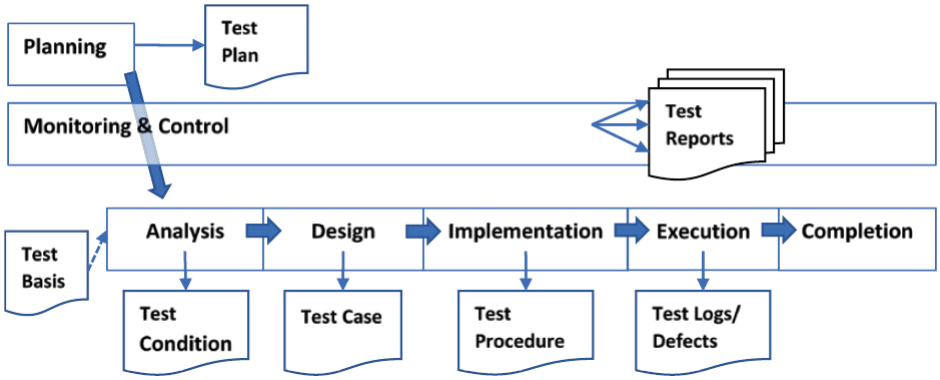

Abbildung 4.3: Testprozess - entnommen aus [66, S. 108]

S. 5] und [35, S. 4] Veränderungen im Druchsatz und der Latenz nutzen, um die Rückschlüsse die Skalierung des System zu ziehen. Weiterhin ist durch das Beauftragen einer Transaktion noch nicht gewährleistet, dass diese auch erfolgreich abgeschlossen wird. Aus diesem Grund wird außerdem die Erfolgsrate betrachtet. Diese berechnet sich aus dem Anteil der Transaktionen, die von den insgesamt durchgeführten Transaktionen erfolgreich durchgeführt wurden [26, S. 2].
Um die Auswirkungen größerer Transaktionen auf das System zu betrachten, wird zudem der Bedarf an Ressourcen erfasst. Zudem soll darüber sichergestellt werden, dass die Testergebnisse nicht durch die Grenzen der Testumgebung verfälscht werden. Wie in der Ausarbeitung [35, S. 4] sollen daher die CPU und RAM Auslastung betrachtet werden. Anstatt die Werte jedoch einzeln zu erfassen, soll der druchschnittliche Bedarf aller Knoten ermittelt werden.

4.5 Planung der Performance-Tests

Der in der Arbeit verwendete Prozess orientiert sich in erster Linie an dem Buch „Performance Testing“ von Keith Yorkston [66], welches sich seinerseits nach dem Lehrplan des International Software Testing Qualifications Board (abgekürzt ISTQB) richtet. Dieses ist ein international anerkanntes Zertifizierungssystem für Software Tests, welches unter anderem eine Spezialisierung für Performance Tests anbietet [37].
Das verwendete Prozessmodell ist in Abbildung 4.3 zu sehen. Die ersten drei Phasen Analyse, Design und Implementierung dienen dazu, die Testfälle zu definieren. In den letzten beiden Phasen werden die erstellten Testfälle ausgeführt und aus den Ergebnissen Empfehlungen abgeleitet. Einige Aspekte dieses Prozesses sind im Rahmen der Arbeit nicht notwendig, da die Performance-Tests bereits in einem relativ spezifischen Rahmen durchgeführt werden. So gibt es beispielsweise keine bestehenden Systeme, reale Nutzer oder Geschäftsprozesse.

4.5.1 Test Basis

Vor der Planung und Erstellung der Tests, werden die Grundlagen erfasst und analysiert. Dazu gehört das Sammeln der Performance Anforderungen, die Betrachtung beteiligter Systeme und ein Verständnis der beteiligten Prozesse.

User Stories

Die User Stories werden entsprechend der Satzschablone von Rupp und Pohl verfasst [58, S. 75]. Entsprechend dieser beschreibt eine Rolle im ersten Punkt, welche Funktion das System erfüllen soll und im zweiten Punkt ihre Motivation hinter der User Story. Zusätzlich werden Akzeptanzkriterien angeführt, welche festlegen, wann eine User Story als umgesetzt gilt. Die User Stories können in dem Fall teilweise aus den Forschungsfragen abgeleitet werden.
Die User Stories sind ein nützliches Werkzeug, da sie Ziele, Motivationen und messbare Kriterien definieren, welche im weiteren Verlauf für die Erstellung der Performance-Tests genutzt werden können.

Funktion des Systems:

Als Tester/Autor

- möchte ich einen Provenance Graphen in Ethereum speichern können,
- sodass er sicher vor Manipulationen ist.

Akzeptanzkriterium: Sobald die Transaktion bearbeitet wurde, kann der Status anhand des Transaktionshashes überprüft werden.

Als Tester/Autor

- möchte ich die Performance für den Konsensalgorithmus Proof-of-Authority bewerten.
- sodass ich Abschätzungen für Netze, welchen diesen Konsensalgorithmus nutzen machen kann.

Akzeptanzkriterium: Es konnten Metriken für die Testdaten unter Verwendung des Konsensalgorithmus PoA erhoben werden.

Als Tester/Autor

- möchte ich die Performance für den Konsensalgorithmus Proof-of-Stake bewerten.
- sodass ich Abschätzungen für Netze, welchen diesen Konsensalgorithmus nutzen tätigen kann.

Akzeptanzkriterium: Es konnten Metriken für die Testdaten unter Verwendung des Konsensalgorithmus PoS erhoben werden.

Durchführung der Performance Tests

Als Tester/Autor

- möchte ich die Skalierung der Performance beim Speichern von Provenance Graphen messen.
- sodass ich die Performance beim Speichern anderer Performance Graphen abschätzen kann.

Akzeptanzkriterium: Aus der Auswertung der erfassten Metriken, geht ein solcher Skalierungsfaktor hervor.

Als Tester/Autor

- möchte ich den Einfluss unterschiedlicher Gas Limits auf die Performance untersuchen.
- sodass ich die Performance beim Speichern großer Provenance-Graphen messen kann.

Akzeptanzkriterium: Das Gas Limit des Netzes wird verändert und es können Werte für bereits getestete Provenance-Graphen erfasst werden.

Als Tester/Autor

- möchte ich den Einfluss unterschiedlich vieler Knoten auf die Performance untersuchen.
- sodass ich weiß, wie sich weitere Teilnehmer auf die Performance auswirken.

Akzeptanzkriterium: Die Anzahl der Knoten im Netz wird verändert und es können Werte für bereits getestete Provenance-Graphen erfasst werden.

Volumetrische Analyse

Ziel der volumetrischen Analyse ist es, möglichst realitätsnahe Modelle für die Nutzung zu identifizieren. Zusammen mit den Anforderungen ergeben sie das Einsatzprofil, welches als Grundlage genutzt wird, um die authentische Nutzung des Systems nachzustellen [66, S. 210f].
Da für die geplanten Performance-Tests weder ein echter Einsatzzweck noch Nutzergruppen existieren, werden hypothetische Szenarien aus den Forschungsfragen abgeleitet. Bei der Erstellung werden zur Orientierung die Fragen aus [66, S. 20ff] verwendet. Da die Modelle zum Erstellen von Lasttests genutzt werden sollen und es nur um die Aufrufe der schreibe-, lese- und update-Funktionen des Smart Contracts geht, konzentrieren sich die Modellbeschreibungen darauf und lassen eventuelle weitere Abläufe aus. Je nach Forschungsfrage ergeben sich unterschiedliche Szenarien für einen hypothetischen Einsatz:

RQ1 - Modell 1: Eine private Blockchain soll genutzt werden, um Provenance-Graphen unterschiedlicher Größe zu speichern. Die Größe der zu speichernden Daten kann dabei variieren. Zu Beginn sind Provenance-Graphen kleiner, können über weitere Transaktionen aber erweitert werden. Geliefert werden die Daten von einem externen System über eine Schnittstelle. Weitere Systeminstanzen, welche von Nutzern bedient werden, greifen auf die gespeicherten Daten in der Blockchain zu, lesen diese aus und verifizieren sie. Anschließend werden die Daten in weiteren Prozessen verarbeitet. Die Nutzer können sich dabei verteilt an jedem erdenklichen Ort befinden. Das Auslesen der Daten verteilt sich hauptsächlich auf die Kernarbeitszeit zwischen 9 und 15 Uhr. Das externe System kann die Daten zu jeder Zeit einspielen.

RQ1 - Modell 2: Mehrere Nutzer greifen über Clients auf eine private Blockchain zu, um Provenance-Graphen zu speichern oder sie auszulesen. Die Zugriffe können dabei zeitgleich und über mehrere Endpunkte geschehen.

Iter.	1	2	3	4	5	6	7	8	9
Größe [KB]	0.5	1	1.5	2	2.5	10	100	500	1000

Tabelle 4.2: Testdaten Größe

RQ2: Die Provenance-Daten sollen möglichst schnell und in möglichst allen Größen innerhalb der Blockchain gespeichert werden können. Die Größe, des Blockchain-Netzes wird nach der Testphase weiter ausgebaut, so dass Abteilungen mit besonders hohen Zugriffszahlen einen eigenen Endpunkt zur Verfügung gestellt bekommen.

4.5.2 Test Design

In [66] werden die Testanalyse und das Testdesign getrennt betrachtet. Hier werden die beiden Abschnitte miteinander verbunden, da es sich beim Design der Szenarien weniger um eine Analyse handelt, als um fest definierte Testszenarien.

Testdaten

Zunächst werden die verwendeten Testdaten beschrieben. Entsprechend [66, S. 122] werden die Testdaten in drei Kategorien unterteilt:
Bei den Masterdaten handelt es sich um Daten, welche bereits vor der Ausführung der Performance-Tests im System enthalten sind. Diese sind der kompilierte Smart Contract als Byte-Satz, die in der Genesis-Datei angelegten Accounts und die Adresse des Smart Contract. Der Besu-Client erzeugt die Accounts mitsamt der zugewiesenen Guthaben beim Start einer neuen Blockchain. Der Smart Contract wird mit einer Transaktion vorbereitend in die Blockchain eingespielt. Als Resultat wird die Adresse des Smart Contract erzeugt.

Benutzerdefinierte Daten werden während der Ausführung von Testfällen eingespielt. Bei den Testdaten handelt es sich um Provenance-Graphen, welche in unterschiedlichen Größen erzeugt werden sollen. Die Grundlage der Datenmenge definieren die mit Hilfe des Provenance-Graph Generators generierten Graphen. Durch die standardmäßig gesetzten Prefixes und die Struktur des Dokuments, entsteht pro Graph ein Offset von etwa 400 Byte. Wird ein Provenance-Graph mit einem Knoten erzeugt, kommen etwa 50 Byte für die Definition des Knotens hinzu und definiert somit eine minimale Datenmenge von ungefähr 450 Byte. Die Datenmenge anstelle der tatsächlichen Graphgröße für die Definition der Testdaten zu nutzen, ist hinsichtlich späterer Abschätzungen sinnvoller, da reale Provenance-Graphen bezüglich ihres Speicherbedarfs pro Kante oder Knoten variieren können. Weiterhin können andere Elemente, wie die Anzahl der verwendeten Prefixes oder Bundles eine zusätzliche Varianz der Datenmenge bedeuten.
Zur Feststellung des Performanzverhaltens, wird ein initiales Datenvolumen von 500 Byte gewählt und für die weiteren Testdaten jeweils um 500 Byte erweitert, bis ein Datenvolumen von 2.5 KB erreicht ist. Auf diese Weise soll der Einfluss kleiner Datenmengen auf die Performance erfasst werden. Sobald eine Datenmenge von 2.5 KB erreicht ist, wird im nächsten Schritt ein Datenvolumen von 10 KB gewählt und pro Iteration um eine Zehnerpotenz erhöht, bis ein Wert von maximal einem Megabyte erreicht ist. Zusätzlich wird für ein später durchgeführte Testszenario ein weiterer Provenance-Graph von 500 KB erstellt. Die vollständige Auflistung der Testdatengrößen ist in Tabelle 4.2 zu sehen.
Transaktionsbezogene Daten werden während der Ausführung erzeugt und stehen mit den

Ergebnissen im Zusammenhang. Insbesondere sind dies die Transaktionshashes, über welche der Status einer eingereichten Transaktion überprüft werden kann.

Konfiguration der Testnetze

Im Zusammenhang mit Forschungsfrage *RQ2*, sollen Testnetze unterschiedlicher Größe betrachtet werden. Die Größe eines Netzes wird über die Anzahl der Knoten festgelegt. Die Variation der Netzgröße ist bereits Bestandteil anderer Arbeiten gewesen und wird als horizontale Skalierung bezeichnet [35, S. 5]. Die Netzgrößen werden abhängig von der Anzahl der signierenden Knoten und des verwendeten Konsensalgorithmus bestimmt. PoA-Netze mit einem Signer sollen mit bis zu 16 Knoten getestet werden. Da realistisch etwa 60 GB Arbeitsspeicher auf der Testumgebung zur Verfügung stehen und alle Netze unter den gleichen Vorraussetzungen betrieben werden sollen, werden jedem Knoten 3 GB RAM zugeweisen.
Im Vergleich dazu sollen PoA-Netze betrachtet werden, in denen jeder Knoten Transaktionen signieren kann. Testläufe haben ergeben, dass die Ressourcen der Testumgebung nur knapp für ein Netz mit acht Knoten, welche signieren können ausreichen. Aus diesem Grund wird auf das Testsetup mit 16 Knoten verzichtet und stattdessen werden maximal acht Knoten genutzt.
Jedes Testnetz wird zur Vereinfachung mit einer eindeutigen Kennung versehen. Verwendet wird hierzu die folgenden Notationen: */Anzahl Knoten/n/Anzahl Signer/s*
Normalerweise fallen für eine Transaktion Gebüren abhängig vom Berechnungsaufwand an. Diese berechnen sich aus dem für die Transaktion genutztem *gas* und dem *gas price* (vgl. [3]). Da es sich um ein privates Netz handelt, benötigen die Teilnehmer keinen Anreiz, um eine Transaktion zu bearbeiten, weshalb der *gas price* auf 0 gesetzt wird. Ebenfalls im Zuge der Forschungsfrage *RQ2*, soll der Einfluss des Gas Limits betrachtet werden. Bei der Definition der Größe ist zu beachten, dass die maximale Testdatengröße von 1 MB weiterhin gespeichert werden können muss. Aus einem Testlauf mit einer Datenmenge von einem Megabyte, ergibt sich eine untere Grenze von ungefähr $750 \cdot 10^6$ Einheiten *gas*. Als obere Grenze wird das Gas Limit aus der Arbeit [35] übernommen, welches etwa $9 \cdot 10^{15}$ Einheiten *gas* entspricht. Um Forschungsfrage *RQ3* beantworten zu können, werden weiter Gas Limits in Höhe von $30 \cdot 10^6$ und $10 \cdot 10^6$ in einer Zusätzlichen Genesis-Datei definiert.

Definition der Workloads

Aus Experimenten mit unterschiedlichen Netzen hat sich ergeben, dass es sinnvoll sein kann verschiedene Workloads zu verwenden. Die Herausforderung besteht darin, die Workloads so zu definieren, dass die Ergebnisse dennoch miteinander vergleichbar bleiben. Der Aufbau der Workloads ähnelt dem der Arbeiten [35], [63] und [64], welche ebenfalls mit dem Benchmark-Tool Hyperledger Caliper [7] gearbeitet haben. In den genannten Arbeiten wird häufig mit dem vordefinierten Testaufbau gearbeitet, welcher aus den Transaktionen „open“, „query“, „transfer“ und dem Smart Contract „simple“ besteht.

Workload *w1* (Latenz): Das Ziel des Workloads ist es, die Latenz für die unterschiedlichen Testdatengrößen zu ermitteln. Für jede Testdatei werden 100 Transaktionen erzeugt, welche sequenziell ausgeführt werden. Die Abstände für die Ausführung werden so gewählt, dass ähnlich wie in den Arbeiten [63, S. 7] und [35, S. 4] eine geringe Last für das Netz entsteht und die Transaktionen ohne Verzögerung bearbeitet werden können. Der Ausführungszeitpunkt soll zwischen der Erzeugung von zwei Blöcken verteilt werden. Aus diesem Grund werden unterschiedliche Senderaten definiert. Die Senderate wird in *tps* angegeben,

was für „transactions per second“ steht. Für die Testdaten 0.5 KB bis 10 KB werden die ersten 20 Transaktionen mit einer Rate von 1.5 tps ausgeführt, die nächsten 20 mit einer Rate von 1 tps und die restlichen 60 mit 1.3 tps. Die Testdaten mit 100 KB werden ebenfalls nach dem Verhältnis 1:1:3 aufgeteilt und mit 0.5, 0.4 und 0.45 tps ausgeführt. Ebenso werden auch die Transaktionen mit der Größe von einem Megabyte aufgeteilt und mit 0.04, 0.02 und 0.03 tps versendet.

Workload *w1-1* (Latenz): Der Workload stellt eine Erweiterung zu *w1* dar. Bei der Ausführung von *w1* weisen die Netze mit steigender Knotenanzahl Unterschiede in der Verarbeitungsgeschwindigkeit, ab einer Testdatengröße von 10 KB auf. Dies schlägt sich insbesondere in einer ansteigenden Latenz nieder, obwohl die Blockchain hinsichtlich zeitgleich offener Transaktionen unter einer geringen Last steht. Wie zuvor auch die kleineren Transaktionen. Beispielhaft ist dies am Report „report_besu_poa_8n_8s_w1.html“ zu sehen. Aus diesem Grund wird ein zweiter Workload für Testdaten ab 10 KB erstellt. Die tps Rate für die Testläufe mit 10 KB, 100 KB und 1 MB, werden hierzu so definiert, dass das Netz *8n8s* möglichst nur eine Transaktion zeitgleich ausführt. Das Verhältnis bei der Aufteilung der Senderaten bleibt mit 1:1:3 bestehen. Der 10 KB große Provenance-Graph wird mit den Raten 0.07, 0.08 und 0.06 tps gesendet. Der Provenance-Graph von 100 KB mit 0.05, 0.03 und 0.04 tps. Die Transaktionen mit der Größe von einem MB, werden mit 0.01, 0.011 und 0.013 tps gesendet. Da außerdem beobachtet werden konnte, dass Hyperledger Caliper zwischenzeitlich Transaktionen schneller ausführt als definiert, wird die Anzahl der Transaktionen pro Runde auf 20 reduziert.

Workload *w1-2* (Latenz): Der Workload stellt eine erneute Erweiterung der vorherigen Workloads dar und definiert Tests für den Testdatensatz mit 500 KB. Die Datenpunkte für diese Testdatengröße werden benötigt, um das Skalierungsverhalten genauer analysieren und beschreiben zu können. Im Rahmen des Tests werden erneut 20 Transaktionen mit einer Senderate von 0.01, 0.011 und 0.013 ausgeführt. Die Aufteilung der Senderaten bleibt mit 1:1:3 bestehen.

Workload *w1-3* (Latenz): Nachdem der Workload *w1-1* die Testdaten ab 10 KB mit einer geringeren Senderate ausführt, definiert dieser Workload geringere Senderaten für die Testdaten zwischen 0.5 KB und 10 KB. Pro Testdatengröße werden 20 Transaktionen gesendet. Die Senderaten sind für alle Testdaten identisch und sind werden mit 0.07, 0.08 und 0.06 definiert. Wie bei den vorherigen Workloads werden diese im Verhältnis 1:1:3 gewichtet. Ziel des Workloads ist es zu prüfen, ob sich die Latenz ändert, wenn auch bei kleineren Transaktionen keine Last entsteht.

Workload *w1-4* (Latenz): Dieser Workload wird insbesondere für das Netz 16n1s erstellt und dient zur Überprüfung des Messergebnisses der Testdatengröße von 1 MB. Dazu wird der Provenance-Graph 60 mal mit den Senderaten 0.01, 0.011 und 0.013 an die Blockchain gesendet. Die Senderaten werden anteilig des Verhältnisses 1:1:3 aufgeteilt. Workload *w2* (Durchsatz): Das Ziel des Workloads ist es, den maximalen Durchsatz für alle Testdatengrößen zu ermitteln, indem unterschiedliche tps Raten getestet werden. Das Vorgehen orientiert sich an dem der Autoren von [64, S. 9]. Diese nutzen eine feste Anzahl an Transaktionen

und erhöhen die tps Rate mit jedem Lauf um 20, bis der Durchsatz den maximalen tps Wert erreicht. In der vorliegenden Arbeit wird eine höhere Anzahl an Transaktionen gewählt, da bei kleineren Transaktionsmengen und höherer tps Rate, Schwankungen in den Ergebnissen mit gleichen Workload festgestellt werden. Die Anzahl der Transaktionen wird für den 0.5 KB Datensatz auf 2000 gesetzt und die tps Raten im Bereich 60 bis 140 untersucht, wobei diese jede Runde um 20 tps erhöht wird. Um zu verhindern, dass größere Netze bei größeren Testdaten überlasten oder sehr lange für die Tests benötigen, wird die Anzahl der Transaktionen an die Testdatengröße angepasst. Als Basis dafür dient der Testlauf mit dem 1 KB Testdatensatz, für welchen 1000 Transaktionen durchgeführt werden. Die Anzahl der Transaktionen für die anderen Testläufe, wird anteilig berechnet, indem die Testdatengröße in KB durch die Ausgangsgröße 1000 geteilt wird. Die Senderaten werden für die unterschiedlichen Netze über mehrere Iterationen angepasst, bis das Netz nicht überlastet wird und ein Maximum bestimmt werden kann. Workload *w2-1*: Dieser Workload stellt eine Ergänzung zum Workload w2 dar. Netze mit mehreren Knoten, sind durch die hohe Anzahl an Transaktionen schnell überlastet. Aus diesem Grund wird der Workload w2-1 auf eine Senderate zwischen 20 tps und 60 tps beschränkt, wobei die Rate bei jedem Durchlauf um 5 tps erhöht wird.

w4:
Um den Gas Bedarf pro Transaktion zu ermitteln und so Forschungsfrage *RQ3* untersuchen zu können, soll jede Testdatengröße in einen einzelnen Block geschrieben werden. Die Senderaten werden dafür entsprechend niedrig definiert. Die Testdaten von 0.5 KB bis 2.5 KB werden mit 0.05 tps und die Testdaten von 10 KB bis zu einem MB werden mit 0.03 tps gesendet. Zur Kontrolle, werden für jede Testdatengröße zwei Transaktionen ausgeführt.

Szenario Design

Szenario s1: Im Rahmen der Forschungsfrage *RQ1*, soll im ersten Schritt die Performance für unterschiedliche Datenmengen untersucht werden. Ziel ist es nach Möglichkeit einen Skalierungsfaktor zu identifizieren, welcher die Performance in Abhängigkeit zur Datenmenge beschreibt.
Das erste Testszenario beschreibt daher eine Form des Skalierungstests. Skaliert wird hierbei die Größe der Testdaten, welche die generierten Provenance-Graphen darstellen. Diese werden jeweils per Transaktion über den Smart Contract in der Blockchain gespeichert. Die Transaktionen werden über einen virtuellen Nutzer in Form eines Accounts ausgeführt. Die Adresse des Accounts und der zugehörige private Schlüssel werden dafür im Vorhinein erzeugt. Verwendet wird das Netz 4n1s. Nach dem Start der Knoten, verbindet diese ein Python Skript miteinander und der Prometheus-Server wird mit der Konfiguration für das Netz gestartet.
Die Testdaten werden gemäß des definierten Workloads *w1* an den Smart Contract gesendet und die durchschnittliche Latenz für jede Testdatengröße bestimmt. In Summe handelt es sich dabei um 700 Transaktionen.
Zur Untersuchung der Forschungsfrage *RQ2* wird das Szenario s1 ebenfalls mit allen anderen definierten Netze durchgeführt.

Szenario s2: Das zweite Szenario beschreibt einen Lasttest, da viele Transaktionen über eine Hohe Senderate an die Blockchain gesendet werden. Hierzu wird der Workload *w2* genutzt. Die Vorbereitung erfolgt dabei auf gleichem Weg, wie in Szenario s1.

Vorbereitung des Monitorings

Das Benchmark-Tool Caliper [7], bietet mehrere Möglichkeiten, um die Tests zu überwachen. Metriken wie die Latenz und der Durchsatz, wie sie in Abschnitt 4.4 definiert wurden, werden automatisch erfasst. Zusätzlich können die Resourcen des Servers überwacht werden. So kann die CPU Auslastung und RAM-Bedarf für einen Prozess über das Process-Modul beobachtet werden. Genutzt wird hierfür die Aggregierungsfunktion *avg* um den Durchschnitt der CPU Auslastung zu ermitteln. Eine weitere und umfänglichere Variante ist die Abfrage der Metriken von einem Prometheus Server. Dieser wird auch von dem Besu-Client unterstützt und stellt einige vordefinierte Metriken über einen Endpunkt zur Verfügung. So kann z.B. der RAM-Bedarf pro Knoten ausgelesen werden.
Damit können alle ausgewählten Metriken aus Abschnitt 4.4 erfasst werden. Zur visuellen Überwachung wird außerdem das Tool Grafana genutzt, welches die erfassten Metriken des Prometheus-Servers grafisch aufbereiten kann.

Kapitel 5

Prototypen

In diesem Kapitel werden die Prototypen implementiert und die geplante Testumgebung aufgesetzt.

5.1 Implementierung des Speicherverfahrens

Für die Implementierung wird die Web IDE Remix genutzt (https://remix.ethereum.org), welche von ethereum.org zur Verfügung gestellt wird. Diese bietet neben vielen unterstützenden Funktionen, wie der Prüfung der Syntax einen Kompiler. Außerdem kann der Smart Contract testweise in eine virtuelle Blockchain eingespielt und getestet werden.
Als Grundlage dient das ausgewählte Speicherverfahren [61], dessen Quellcode auf Github verfügbar ist. Der Smart Contract ist für ein anderes Provenance Modell speziell für IoT Daten konzipiert. Zudem soll in der Arbeit nur ein ein Provenance Dokument gespeichert werden, welches keine Verweise auf andere Dokumente enthält. Stattdessen ist wichtig, dass das der gespeicherte Provenance-Graph auf Richtigkeit überprüft werden kann. Daher soll der Hash-Wert gespeichert werden können. Im Zuge dessen wird die Struktur *ProvenanceRecord* wie folgt angepasst:

```
struct ProvenanceRecord {
    string hash;
    string context;
    uint index;
}
```

Durch die Änderung der Struktur müssen auch alle Stellen an denen diese verwendet wird überarbeitet werden. Beispielsweise in der Funktion *getProvenance*. Eine wichtige Änderung stellt außerdem die Vergabe eines automatischen Indexes dar, da Hyperledger Caliper keine Möglichkeit bietet den Index des Vorgängers zu speichern, wodurch es zu Fehlern bei der automatischen Ausführung kommen würde. Die Änderung findet in der Funktion „createProvenance“ statt. Die Neuerung beinhaltet das Auslesen des letzten gespeicherten Indexes sowie die Erhöhung von diesem um eins. Außerdem wird die Prüfung eines bereits bestehenden Eintrages auskommentiert. Außerdem werden nicht benötigte Funktionen und Events entfernt. Der gesamte Smart Contract ist im Anhang zu finden.

```
function createProvenance(string memory _hash, string memory
  ↪ _context) public returns (uint index) {
      //require(!isStored(_provId), "Record already exists");
      uint _provId = 0;
```

```
        if (provenanceIndex.length != 0) {
            _provId = provenanceIndex[provenanceIndex.length-1] +
                ↪ 1;
        }

        provenanceIndex.push(_provId);

        provenanceRecords[_provId].hash = _hash;
        provenanceRecords[_provId].context = _context;
        provenanceRecords[_provId].index = provenanceIndex.length -
            ↪ 1;

        return provenanceRecords[_provId].index;
    }
```

Nach der Fertigstellung, wird der Smart Contract mit der solc kompiliert, um die benötigte Byte-Datei zu erzeugen. Solc kann über npm und den Befehl „npm install -g solc“ installiert werden. In der Arbeit wird die Version 0.8.25+commit.b61c2a91.Emscripten.clang genutzt, da das Etherum Netz in der Version Paris genutzt wird und nicht die Operation *PUSH0* unterstützt, welche andernfalls Verwendung finden würde.

5.2 Implementierung des Datengenerators

Die gesamte Implementierung orientiert sich am erstellten Konzept aus Abschnitt 4.1.3. Als Entwicklungsumgebung wird Visual Studio Code v1.88.1 sowie Python 3.7.4 genutzt.
Vorbereitend müssen die Python Packages *prov*, *graphviz* und *pydot* über den Package-Installer pip installiert werden. Zusätzlich wird die Installation von Graphviz vorausgesetzt, dessen *bin*-Verzeichnis zur Umgebungsvariable PATH hinzugefügt werden muss.

main.py:
Die Konfiguration und Ausführung des Datengenerators erfolgt über die Eigenschaften der Klasse *MainProgram*. Der Dateiname, die Anzahl der Knoten, die Tiefe (stages) und die Breite des Graphen, können über den Konstruktor gesetzt werden. Die Klasse stellt ausschließlich die Methode *generate_and_save_graph_as* bereit, welche alle notwendigen Schritte zum Erzeugen und Speichern des Graphen ausführt. Als Parameter können die gewünschten Dateiformate gesetzt werden, in welchen der Graph gespeichert werden soll. Unterstützt werden die Formate PROVN, JSON, PNG, GV und PDF.
Der Graph wird über die Methode *gen_graph* der Klasse *StructureGenerator* erzeugt. Dieser werden die Anzahl der Knoten sowie die Tiefe und Breite des zu erzeugenden Graphen übergeben. Als Ergebnis liefert die Funktion zwei Listen. Die erste Liste enthält für jede Stage eine weitere Liste, welche jeweils die Knoten dieser Stage beinhaltet. Die zweite Liste enthält ebenfalls Listen, welche die Kanten gruppieren, die zwei Stages miteinander verbinden.
Wenn die Generierung der Graph-Struktur erfolgreich war, wird diese mit der Klasse *ProvGenerator* in ein Provenance-Modell überführt. Das Modell wird anschließend abhängig von den zuvor gesetzten Parametern im PROVN und JSON Format gespeichert.
Zum Erzeugen der Visualisierung wird die Klasse *GraphPlotter* genutzt. Sie agiert unabhänig von den Ergebnissen der ProvGenerator-Klasse und verwendet die ursprünglich erzeugten

Listen mit den Knoten und Kanten des Graphen. Abhängig von den gesetzten Parametern, wird der Graph mit Hilfe dieser Klasse als GV, PDF oder PNG gespeichert. Eine Besonderheit bildet dabei die GV-Datei, da es sich bei diese um keine direkte Visualisierung des Graphen handelt. Sie dient als Grundlage bei der Erzeugung aller Ausgabeformate mit Graphviz und somit auch der PDF-Datei. Aus Gründen der Transparenz erfolgt die Generierung GV-Datei dennoch nicht automatisch vor der Erzeugung der PDF-Datei.

node.py:
Die Datei enthält die gleichnamige Klasse *Node*, sowie die Enums *NodeType*, *NodeProvType* und *NodePrefix*, welche alle in der Node-Klasse Verwendung finden. Zusätzlich enthält die Node-Klasse als Attribut eine ID, welche automatisch aufsteigend generiert wird, einen Namen und die Stage in welcher der Knoten liegt.

edge.py:
structure_generator.py:
Über den *StructureGenerator* kann über einen objektorientierten Ansatz ein Graph erzeugt werden. Verwendung finden dafür die Klassen *Node* und *Edge*. Die Generierung wird über die Methode *gen_graph* gestartet, welcher dafür die Anzahl der Knoten, die Tiefe des Graphen (stages), sowie die maximale Breite übergeben werden muss. Repräsentiert wird der zu erstellende Graph von zwei Listen, wobei eine die Knoten und eine die Kanten enthält. Da die Anzahl der Stages bereits bekannt ist, werden beide Listen mit leeren Listen initialisiert. Anschließend wird die Verteilung generiert. Dies ist eine Liste, welche die Anzahl der zu erzeugenden Knoten pro Stage enthält. Generiert wird diese von der Methode *generate_distribution*. Ein Graph fängt dabei immer mit einem Knoten an, weshalb der erste Eintrag immer eine Eins ist. Im Anschluss werden die Knoten auf die Stages verteilt. Dafür wird ab der zweiten Stage eine zufällige Zahl zwischen Eins und der maximalen Breite des Graphen gewählt. Ist die gewählte Zahl größer als die Anzahl der noch zu verteilenden Knoten oder es könnten anschließend nicht mehr ausreichend viele Knoten auf die verbleibenden Stages verteilte werden, wird eine neue Zahl generiert. Andernfalls wird der Wert in die Ergebnisliste übernommen und er Ablauf wiederholt sich für die nächste Stage. Eine weitere Ausnahme bildet die letzte Stage, da in dieser aller Knoten verteilt sein müssen, aber gleichzeitig die maximale Breite nicht überschritten sein darf. Falls sie überschritten werden würde, wird eine zufällige vorherige Stage ausgewählt und dieser wiederum eine zufällige Anzahl an Knoten zugeteilt, ohne dass auch hier die maximale Breite überschritten wird. Der Prozess wiederholt sich so lange, bis die letzte Stage ohne Überschreitung generiert werden kann.
Im nächsten Schritt erfolgt die Generierung des Graphen. Der erste Knoten wird von einem zufällig bestimmten Knotentyp erzeugt. Ab der zweiten Stage werden zuerst die Kanten über die Methode *choose_relation_types* generiert. Da die Typen der Kanten von den Ausgangsknoten abhängig ist, werden diese zusammen mit der Anzahl der zu generierenden Kanten an die Methode übergeben. Für jede zu erzeugende Kante wird als erstes ein zufällige Startknoten aus der vorherigen Stage bestimmt. Abhängig von den Typen der Knoten, werden die mögliche Kantentypen auf Basis von zuvor gesetzten Wahrscheinlichkeiten ermittelt. Diese sind in der Variable *prob_relations_dict* mit dem Wert 0.1 inititalisiert, wodurch sie mit der gleichen Wahrscheinlichkeit ausgewählt werden. Für jeden Knotentyp ist in einer separaten Methode hinterlegt, welche Kantentypen möglich sind. Für die Kantentypen werden die hinterlegten Wahrscheinlichkeit ausgelesen und basierend darauf einer der möglichen Kantentypen ausgewählt. Die ausgewählten Kantentypen werden zusammen mit

ihrem Startknoten in einer Ergebnisliste gespeichert und zurückgegeben.
Abschließend werden die Zielknoten generiert. Für diese ist ebenfalls der Kantentyp entscheident, da die Kantentypen in vielen Fällen nur für einen Zielknotentyp definiert sind. In der Methode *choose_relation_types* sind diese Zuordnungen hinterlegt. Für den Fall, dass mehrere Zielknotentypen möglich sind, wird zufällig einer ausgewählt. Nach der Ermittlung der Standardprefixes für die Zielknoten, werden diese schließlich erstellt und in die aktuelle Stage der Ergebnisliste aufgenommen. Gleiches erfolgt für die generierten Kanten, welchen zusätzlich die Start- und Zielknoten zugewiesen werden. Die redundante Speicherung der Knoten ist nicht notwendig, vereinfacht aber den weiteren Prozess, da Knoten und Kanten meist getrennt voneinander behandelt werden. Wurde der Generierungsprozess für alle Stages durchlaufen, ist die Erstellung der Graphstruktur abgeschlossen.

prov_generator.py:
Die Klasse *ProvGenerator* ist für Überführung in das Provenance Datenmodell von W3C [54] zuständig. Grundlage dafür ist die Klasse *ProvDocument* aus dem *prov*-Package, welches eine Implementierung des Standards für Python zur Verfügung stellt. Eine Instanz dieser Klasse wird über den Konstruktor des *ProvGenerators* angelegt. Zusätzlich erhält dieser den generierten Graphen in Form der zuvor generierten Knoten- und Kantenliste.
Die Methode *set_default_dlr_namespace* fügt dem *ProvDocument* exemplarische Namespaces des DLR hinzu, um diese später für die Knoten verwenden zu können. Die Überführung des Graphen erfolgt über die Methode *map_graph_to_document*. Diese iteriert zunächst über alle Knoten und legt diese abhängig vom Knotentyp als PROV-DM Typ im *ProvDocument* an. Gesetzt werden dafür der Name und der Typ. Für die *activity* wird zusätzlich ein Start- und Endzeitpunkt angegeben.
Das Anlegen der Relationen erfolgt auf ähnliche Weise, indem über alle Kanten iteriert wird. Hier wird der Kantentyp genutzt, um die passende PROV-DM Relation im *ProvDocument* anzulegen. In allen Fällen werden als Attribute die Bezeichner des Start- und Endknotens gesetzt.
Nach Abschluss der Überführung des Graphen in das *ProvDocument*, kann aus diesem über bereitgestellte Methoden ein PROVN oder JSON String erzeugt werden, welcher wiederum über die Methoden *generate_provn* und *generate_json* ausgelesen werden kann.

graph_plotter.py:
Die Klasse *GraphPlotter* kann die generierte Graph-Struktur in unterschiedlichen Formaten visualisieren. Dazu nutzt sie das Python Package *graphviz*. Zusätzlich wird erneut das *prov*-Package genutzt, um den Graphen als PNG zu speichern.
Die Erstellung und Formatierung der GV-Datei ist in weiten Teilen von [25] übernommen und für die Graph-Objekte angepasst worden. Der Aufbau der Datei kann in vier Bereiche unterteilt werden. Zu Beginn erfolgt die Definition des Graph-Types sowie des Knoten-Layouts. Beides ist aus den Zeilen 186 bis 189 aus der Datei *graph_gen.py* von [25] übernommen. Nach der Definition der Knoten erfolgt ein weitere Konfiguration des Layouts, welche aus den Zeilen 197 und 198 übernommen worden ist. Diese beschreiben die Ausrichtung des Graphen und die Darstellung der Kanten. Der nächste Bereich, in welchem die Knoten definiert werden, ist leicht vom Inhalt der Zeilen 192 bis 194 abgewandelt. [25] iteriert über die Knoten und weist diesen ein Label zu. Für die vorliegende Arbeit wird der Prefix des Labels verändert und der Bezeichner durch die Knoten-ID ersetzt. Zusätzlich wird die Liste *id_rank_list* angelegt, welche die Stages repräsentiert und die jeweiligen Knoten-IDs enthält. Diese wird im letzten Abschnitt genutzt, um die Anordnung der Knoten zu bestimmen. Die

FILENAME	MAX_NODES	STAGES	MAX_WIDTH
kb_0.5	1	1	1
kb_1	7	4	3
kb_1.5	12	5	4
kb_2	17	5	4
kb_2.5	21	7	5
kb_10	96	14	10
kb_100	960	120	100
kb_500	4800	3000	1000
mb_1	9600	1200	1000
mb_10	93000	2200	3000
mb_100	910000	300000	150000

Tabelle 5.1: Liste verwendeter Parameter zur Erzeugung der Testdaten

Autoren von [25] lösen dies in den Zeilen 221 bis 229 in leicht abgewandelter Form. Die Definition der Kanten stellt den dritten Abschnitt dar und ist aus den Zeilen 214 bis 217 übernommen und an das Kanten-Objekt angepasst.
Die GV-Datei kann im Anschluss genutzt werden, um den Graphen in viele unterschiedliche Dateiformate zu überführen. Die vollständige Liste der Formate ist unter https://graphviz.org/docs/outputs/ zu finden. Der *GraphPlotter* stellt über die Methode *save_gv_as_pdf* die Speichermöglichkeit im PDF-Format zur Verfügung. Eine andere Möglichkeit bietet die Methode *generate_png*, welche den Graphen gemäß der PROV-DM Notation [54] erzeugt.

5.3 Generierung der Testdaten

Die Testdaten werden entsprechend der geplanten Größen aus dem Konzept erzeugt. Zusätzlich werden auch Provenace-Graphen mit der Größe von 10 MB und 100 MB erzeugt, da Graphen der Größe zu Beginn der Arbeit ebenfalls geplant waren. Außerdem sollen die erzeugten Graphen anhand der definierten Anforderungen überprüft werden. Die richtige Anzahl an Knoten muss experimentell ermittelt werden. Eine Liste mit den gewählten Parametern ist in Tabelle 5.1 zu finden.
Die Graphen zur Prüfung der Anforderungen sind im Ordner „generated_graphs/validate“ zu finden.

Die Anforderungen *F1*, *F2* und *F3* werden automatisch bei der Erstellung des Graphen erfüllt. Über eine manuelle Prüfung kann bestätigt werden, dass alle definierten Relationen aus *F4* verwendet werden. Ebenso wird das Vorkommen der Knotentypen aus Anforderung *F5* geprüft. Die Anforderung *F9* konnte bereits bei der Erzeugung der Testdaten erfüllt werden. Insgesamt werden fünf Provenance-Graphen zur Überprüfung erzeugt und anhand dieser die verbleibenden funktionalen Anforderungen überprüft. Für die Erzeugung von 1000 Knoten im Rahmen von *Q1* benötigt der Graph weniger als eine Sekunde.
Ein beispielhaft generierter Graph ist in Abbildung 5.1 zu sehen. Dieser wurde mit den mit 8 Knoten, 3 Stages und einer maximalen Breite von 5 Knoten generiert. Die dazugehörige PROVN-Datei ist im Anhang zu finden.

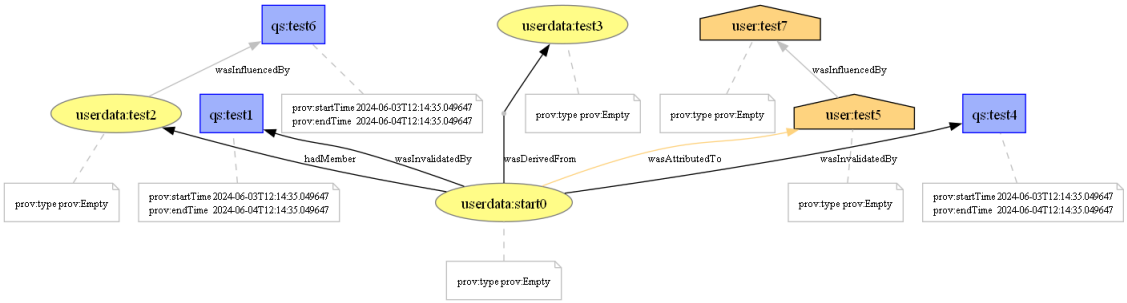

Abbildung 5.1: Veranschaulichung eines generierten Provenance-Graphen

5.4 Aufbau des Testnetzes

Im diesem Abschnitt wird die Umsetzung der geplanten Testnetze beschrieben. Diese basiert auf dem Client Hyperledger Besu und dem Benchmark Tool Hyperledger Caliper.

5.4.1 PoA - Hyperledger Besu

Das Besu Netzwerk wird entsprechend der offiziellen Anleitung [4] aufgebaut. Vorbereitend wird Hyperledger Besu in der aktuellsten Version 24.3.3 aus dem Github Repository heruntergeladen und zusammen mit dem Java OpenJDK 19.0.2 installiert.
Nach der Installation wird das Hauptverzeichnis für das gesamte Netz angelegt. In diesem wird für jeden der geplanten Knoten ein Verzeichnis angelegt, in welchem die Daten als auch der private und öffentliche Schlüssel gespeichert werden. Für den ersten Knoten erfolgt die Erzeugung des Schlüsselpaares exemplarisch mit dem Befehl: *besu –data-path=node1/data public-key export-address –to=node1/data/node1Address*. Dieser Befehl wird für die anderen sieben Knoten angepasst und erneut im Hauptverzeichnis ausgeführt.
Grundlage des Netzes bildet die Genesis-Datei, welche im JSON Format angelegt wird. Diese wird aus der Vorgabe von [4] übernommen und angepasst. Die *chainId* wird auf den Wert 123456 geändert. Entsprechend der Planung in Abschnitt 4.5.2, werden wird das „gasLimit“ auf den Wert „0x1fffffffffff“ und das „contractSizeLimit“ auf das Maximum von 2147483647 gesetzt. Der initiale Signer wird in dem Feld „extraData“ angegeben. Hierfür soll der erste Knoten genutzt werden. Dementsprechend wird dessen öffentlicher Schlüssel aus der Datei „node1Address“ anstelle des Platzhalters eingetragen. Im Anschluss wird ein Account für jeden Knoten in dem Eintrag „alloc“ angelegt. Der Eintrag für den Account des ersten Knotens ist in Listing 5.1 zu sehen.

```
"c7ab6e726d23293eb6581dbc435d464e290019a4": {
        "privateKey": "e5a5d5971acfc08668707
            ↪ b4c292132cc2c3d4d4c7625977ad9f09706f25f8df2",
        "balance": "900000000000000000000000"
    }
```

Listing 5.1: Node 1 - Account

Als Schlüssel für das Account-Objekt wird der öffentliche Schlüssel des Knotens ohne den Hex-Prefix „0x“ genutzt. In dem Feld „privateKey“ wird der private Schlüssel aus der Key-Datei des Knotens ebenfalls ohne Hex-Prefix eingetragen. Abschließend wird die „balance“

mit einem ausreichend hohen Wert zur Ausführung aller Tests angegeben. Der gesamte Inhalt der Genesis-Datei ist im Anhang zu finden.
Um die Knoten der Blockchain zu starten und individuell zu konfigurieren, wird ein Bash-Skript für jeden Knoten erstellt. In der ersten Zeile wird der maximal freigegebene RAM über die Umgebungsvariable „BESU_OPTS“ auf 3 GB limitiert. Anschließend wird der Knoten über den Befehl *besu* gestartet und über die Angabe von Parametern konfiguriert. Das Arbeitsverzeichnis de Knotens wird über „data-path“ angegeben und auf die zuvor erstellten Ordner gesetzt. Die Genesis-Datei kann über den „genesis-file“ angegeben werden und ist insbesondere zur Initialisierung der Blockchain notwendig. Da alle Knoten miteinander kommunizieren können sollen, wird die „network-id“ entsprechend der Dokumentation [9] auf den selben Wert wie die Chain-ID gesetzt. Über die Optionen „host-allowlist“ und „rpc-http-cors-origins“ wird die Verbindung mit Knoten aus allen Netzen grundlegend erlaubt. Das automatische Suchen von Knoten aber mit „discovery-enabled“ deaktiviert. Um die Verbindungen zwischen den Knoten kontrolliert anzulegen, werden die RPC-HTTP Endpunkte der Knoten freigeschaltet und mit unterschiedlichen Ports belegt. Zusätzlich zu den Standard-APIs, werden „ADMIN“ und „TXPOOL“ freigegeben. Dadurch ist es später möglich die Verbidungen zwischen den Knoten zu verändern und den Transaktionspool zu prüfen. Analog dazu werden auch die RPC-WebSockets konfiguriert, welche für die Ausführung der Performance-Tests genutzt werden. Um die Abfrage der knoteneigenen Metriken über Prometheus zugänglich zu machen, werden diese ebenfalls aktiviert und über separate Ports freigegeben. Der minimale Gas-Preis wird gemäß der Dokumentation [3] auf 0 gesetzt, welches den letzten Schritt bei der Konfiguration eines Netzes ohne Gas darstellt. Insbesondere für Tests mit großen Transaktionen, sind die Parameter „rpc-tx-feecap“, „poa-block-txs-selection-max-time“ und „tx-pool-max-future-by-sender“ von Bedeutung. Das „rpc-tx-feecap“ definiert eine Gebührengrenze, welche standardmäßig die Ausführung von teureren und damit größeren Transaktionen verhindert. Über den Wert 0 kann diese deaktiviert werden. Um Timeouts bei der Ausführung großer Transaktionen zu verhindern, kann über den Parameter „poa-block-txs-selection-max-time“ prozentual angegeben werden, wie viel Zeit bei der Erzeugung eines Blocks, für die Auswahl von Transaktionen genutzt werden darf. Mit Angabe des Wertes von +1000 wird sichergestellt, dass Transaktionen aus diesem Grund nicht abgebrochen werden. Abschließend wird mit der Option „tx-pool-max-future-by-sender“ die Anzahl erlaubter Transaktionen in einem Block pro Auftraggeber auf 5000 erhöht. Dies ist besonders bei der zeitgleichen Ausführung vieler kleiner Transaktionen von Bedeutung. Weitere Informationen zu den genannten Parametern sind in der Dokumentation zu finden ([1] und [11]).
Abschließend werden die Knoten entsprechend der Dokumentation [13] statisch miteinander verbunden. Hierzu werden alle Knoten einmal gestartet, wodurch eine enode-URL generiert wird. Die enodes der Knoten werden dann in der Datei „static-nodes.json“ im Verzeichnis der Knoten aufgelistet, um initiale Verbindungen herzustellen.

5.4.2 Hilfsskripte

Um den Aufwand beim Aufbauen und Zurücksetzen der Testnetze zu reduzieren, werden unterschiedliche Hilfsskripte erstellt. Alle beschriebenen Skripte sind im Anhang der Arbeit zu finden.

clean.sh:
Um die Testnetze nach jedem Testlauf zurücksetzen zu können, werden Bash-Skripte erstellt, welche die Datenbanken und Caches aller Knoten löschen. Ein solches Skript wird für alle aufgebauten Netze erstellt.

besu_propose_signers.py:
Um einen Signer zu einem Netz hinzuzufügen, muss dieser von der Mehrheit der vorhandenen Signer vorgeschlagen werden [2]. Möglich ist dies über die Methode „clique_propose“, welche über den HTTP-Endpunkt eines Knotens aufgerufen werden kann. Dieser kann der öffentliche Schlüssel des neuen Signers zusammen mit der Abstimmung übergeben werden.
Das Skript ist in Python geschrieben und nutzt neben der genannten Methode ebenso die Methode „clique_getSigners“. Mit dieser können alle authorisierten Signer eines Netzes abgefragt werden. Beide Methoden können über die REST Methode POST aufgerufen werden. Zur Umsetzung wird das request-Package für Python verwendet. Über die Kommandozeile kann dem Skript die Anzahl der Knoten übergeben werden, welche zum Signieren befähigt werden sollen. Anschließend wird eine Schleife solange im 15 Sekunden Takt durchlaufen, bis die Anzahl der Signer mit der Anzahl der übergebenen Knoten übereinstimmt. Solange dies nicht der Fall ist, werden die Signer mit der Methode „get_signers“ abgefragt. Diese nutzt das request-Package, um eine Liste mit allen Signern vom ersten Knoten abzufragen. Deren Einträge werden anschließend aus einer Liste mit den Adressen aller potenziellen Signer entfernt, wodurch nur noch die Adressen der fehlenden Knoten übrig bleiben. Diese noch nicht authorisierten Signer werden anschließend mit der Methode „propose_signers“ an alle Knoten gesendet. Im Anschluss wiederholt sich der Vorgang solange, bis alle geforderten Knoten signieren können.

5.4.3 Hyperledger Caliper

Als Grundlage des Frameworks wird zu Beginn Node.js in der Version 16.20.2 installiert. Als der Dokumentation geht zwar hervor, dass Node.js in Version 10 vorausgesetzt ist, in dem Changelog von Hyperledger Caliper v0.5.0 ist aber vermerkt, dass auch Node.js v16 unterstützt wird. Im Anschluss wird das Tool [7] von Github geklont und in dessen Verzeichnis per NPM installiert. Zum Zeitpunkt der Testausführung ist die neuste verfügbare Version v0.5.0, welche daher Verwendung findet. Bevor Hyperledger Caliper eingesetzt werden kann, muss das zu testende System (SUT) über den „bind“-Befehl eingebunden werden. Zusammengesetzt wird dieser aus dem Namen des SUT und eines bereitgestellten Software Developer Kits (SDK). Eine Liste mit allen unterstützten Kombinationen aus SUT und SDK ist in der Dokumentation [8] unter „Installing and Running Caliper“ zu finden.
Vor der Einbindung und dem Start des Tools, müssen noch ein Testszenario und eine Netzkonfiguration angegeben werden. Da mit Hilfe von Hyperledger Caliper sowohl Geth- als auch Besu-Netze getestet werden sollen, müssen für beide Netzarten Konfigurationen erstellt werden. Die Erstellung dieser orientiert sich dabei an der an der „Connector Configuration“ für Ethereum in der Dokumentation [8]. Für beide wird als „blockchain“ der Wert „ethereum“ eingetragen, nur die Konfiguration von „ethereum“ unterscheidet sich geringfügig. Als erstes wird für beide der WebSocket-Endpunkt des ersten Knotens als „url“ eingetragen. Anschließend muss die Adresse des Accounts angegeben werden, welcher den Smart Contract einspielt sowie eine Information, um Transaktionen mit diesem ausführen zu können. Der Account wird im Feld „contractDeployerAddress“ und „fromAddress“ eingetragen. Der Zugang zu diesem wird für Besu über den privaten Schlüssel in den Feldern „contractDeployerAd-

dressPrivateKey“ und „fromAddressPrivateKey“ angegeben. Für Geth wird das Passwort in den Feldern „contractDeployerAddressPassword“ und „fromAddressPassword“ hinterlegt. Für beide werden die Daten des ersten Accounts genutzt. Die verbleibenden Felder und Werte sind für beide Konfigurationen gleich. Das Feld „transactionConfirmationBlocks“ definiert, ab wie vielen Folge-Transaktionen eine zuvor ausgeführte Transaktion als sicher gilt. Hier wird der Standardwert 2 beibehalten.
Abschließend folgt die Konfiguration des Smart Contract, welcher zu Beginn eines jeden Testszenarios eingespielt wird. Vorher muss dieser in einer sparaten JSON-Datei definiert werden. In dieser wird der Name angegeben, über welchen der Smart Contract später aufgerufen werden kann. Dann folgt der Inhalt der ABI-Datei und der kompilierte Bytecode. Zuletzt wird festgelegt, wie viel Gas benötigt wird, um den Smart Contract einzuspielen.
In der Netzkonfiguration kann dieser Smart Contract angelegt werden. Als Objekt-Schlüssel wird der zuvor hinterlegt Name „provStorageAutoIndex“ angegeben. Über „path“ muss der Pfad zur JSON-Datei angegeben werden. In dem Feld „Gas“, können optional Gas-Limits für den Aufruf der jeweiligen Funktionen des Smart Contract hinterlegt werden. Auf diese Weise muss der Knoten das Gas für den Aufruf nicht berechnen, was durch den erhöhten Kommunikationsaufwand eine höhere Latenz beim Ausführen der Transaktion zur Folge hätte. Aus diesem Grund wird zum Schreiben mit der Funktion „createProvenance“ ein Wert von 1200000000000 und zum Lesen mit der Funktion „getProvenance“ ein Wert von 100000000 hinterlegt. Auf Basis zuvor durchgeführter manueller Tests ist bekannt, dass beide Werte mit Abstand über dem tatsächlichen Gas-Bedarf liegen.
Alle beschrieben Konfigurationen aus dem Abschnitt sind im Anhang der Arbeit zu finden.

Kapitel 6

Evaluation

6.1 Auswertung der Performance Tests

Alle erzeugten Reports und Übersichten in Form von Excel-Datein sind im angefügte Verzeichnis „reports“ der Arbeit zu finden. Bei den Performance-Tests wird der Fokus auf die horizontale Skalierung durch die Veränderung der Netzgrößen gelegt. Dennoch skalieren die Ressourcen im Hinblick auf den verfügbaren RAM mit, da dieser pro Konten zugewiesen wird. Die verwendeten Grafiken werden in vielen Fällen mit zwei unterschiedlichen Skalierungen abgebildet, um möglichst viele Datenpunkte ablesbar zu visualisieren.

6.1.1 Proof of Authority - Hyperledger Besu

Die folgenden Auswertungen beziehen sich auf die Performance-Tests, welche mit Testnetzen auf Basis des Ethereum-Clients Hyperledger Besu durchgeführt wurden.

Latenz

Tabelle 1 im Anhang der Arbeit zeigt die Messergebnisse der Latenz-Tests für einen signierenden Knoten. Die Werte für die einzelnen Netze sind in Sekunden angegeben. Die Anzahl der jeweils durchgeführten Transaktionen ist in dem Feld „TX“ vermerkt. Trotz der reduzierten Anzahl an Transaktionen in den Workloads, variieren die Ergebnisse nach mehrfacher Ausführung wenig. Auch im Vergleich zwischen den Testnetzen scheinen die Ergebnisse plausibel. Dennoch ist anzumerken, dass die Datengrundlage mit 20 Transaktionen pro Testdatengröße geringer als in anderen Ausarbeitungen ausfällt (vgl. [60], [35], [64], [63]). Der Workload *w1* orientiert sich mit 100 Transaktionen pro Testdatengröße stärker an den genannten Arbeiten und nutzt ebenso wie diese höhere Senderaten. Dadurch kommt es zu mehreren offenen Transaktionen im Transaktionspool. Die Anzahl der offenen Transaktionen ist bei den kleineren Testdatengrößen höher und liegt zwischen 4 und 15. Bei den Transaktionen ab 100 KB, wird die Anzahl auf 1 - 4 offene Transaktionen reduziert, da die Threads der Knoten sonst Gefahr laufen zu blockieren. Im Vergleich mit den Workloads *w1-1* und *w1-3*, welche gemeinsam ebenfalls alle Testdatengrößen abdecken, fällt auf, dass die Latenz für die Testdatengrößen von 0.5 KB bis 10 KB ähnlich hoch ist. Die gemessene Latenz für die Testdatengrößen von 100 KB und 1 MB fällt deutlich höher aus. Eine Gegenüberstellung der Netze 1n1s und 16n1s ist in Abbildung 6.1 zu sehen. Für manche Netze scheint es einen Zusammenhang mit den Auslastungsspitzen der CPU zu geben, welcher jedoch nicht auf das Netz 16n1s zutrifft. Zusätzlich sind weder die durchschnittliche CPU Auslastung noch der RAM-Bedarf auffallend hoch. Zur Überprüfung des Messergebnisses dient der Workload *w1-4*, welcher 60 Transaktionen mit den

	1n1s (w1)		1n1s (w1-1 + 1-3)		16n1s (w1)		16n1s (w1-1 + 1-3)	
Größe [KB]	**Latenz [s]**	**max. CPU [%]**	**Latenz [s]**	**max. CPU [%]**	**Latenz [s]**	**max. CPU [%]**	**Latenz [s]**	**max. CPU [%]**
0,5	7,89	39,33	7,12	16	7,64	37,33	7,28	6,33
1	7,46	34,23	7,84	16,89	7,5	24,28	7,7	5,26
1,5	7,43	24	7,91	8,94	7,63	22,04	7,63	14,02
2	7,52	25,33	7,86	6,71	7,63	18,15	7,73	3,75
2,5	7,59	26	7,93	5,96	7,74	22,24	7,73	5,02
10	8,34	84	7,28	35	9,19	48,97	7,97	8,23
100	11,88	237,67	8,77	81	16,43	75,25	8,86	46,82
1000	16,33	268,46	14,94	252,67	25,22	75,41	19,36	74,47

Tabelle 6.1: Latenz - Gegenüberstellung von *w1* und *w1-1 + w1-3*

niedrigen Senderaten von *w1-1* ausführt. Dieser bestätigt mit einer durchschnittlichen Latenz von 20.33 s die gemessene Latenz von *w1-1*.
Die Autoren Wang et al. messen in [64, S. 8] einen leichten Anstieg der Latenz durch die Erhöhung der Senderate. Aufgrund der zuvor beschriebenen Messungen besteht die Vermutung, dass zusätzlich die Größe der zeitgleich offenen Transaktionen einen Einfluss auf die Erhöhung der Latenz hat und dieser mit der Transaktionsgröße zunimmt. Im Zuge der Performance-Tests mit *w1* betrifft dies Transaktionen ab einer Größe von 100 KB. Infolgedessen werden für die weiteren Auswertungen die Ergebnisse von *w1-1* und *w1-3* genutzt.
Aus Abbildung 6.1 lässt sich entnehmen, dass die Anzahl der Knoten bei einem Signer pro Netz, einen geringen Einfluss auf die Latenz hat. Die Daten der Grafiken basieren auf Tabelle 1. Dieser ist zu entnehmen, dass sich die Latenz des Netzes 16n1s im Vergleich zum Netz 1n1s, beim Speichern eines Datensatzes von einem MB um 18.5% erhöht. Das Skalierungsverhalten unterscheidet sich leicht für jede Netzgröße. Einheitlich kann dieses aber als linear beschrieben werden. Eine untere Grenze bildet in dem Fall das Netz 2n1n, wobei dies auch durch Abweichungen in den Messungen zustande kommen kann. Die Abweichung bei einer Testdatengröße von 1 MB beträgt nur 0.09 s im Vergleich zum Messergebnis des Netzes 1n1s. Die Skalierungsfunktion wird aus den Datenpunkten für 1 KB und 1 MB berechnet und kann mit der Gleichung $f(x) = 0.006967x + 7.883$ beschrieben werden. Die größte Anstieg wird für das Netz 16n1s beobachtet und kann mit der Funktion $f(x) = 0.01167x + 7,6883$ beschrieben werden, deren Grundlage ebenfalls die Datenpunkte für 1 KB und 1 MB liefern. Anzumerken ist, dass sich die Grenzen nur auf die betrachten Netzgrößen beziehen und sich durch das hinzufügen von Knoten nach oben verschieben kann.
Abbildung 6.2 zeigt die Latenz in Abhängigkeit der Anzahl der Knoten für Netze in denen alle Knoten signieren können. Den Graphen liegen die Daten der Tabelle 2 zugrunde. Auch für diese Netze lässt sich der Anstieg der Latenz linear beschreiben. Der Zuwachs nimmt mit der Anzahl der signierenden Knoten weiter zu. Am stärksten fällt dies demnach bei der Betrachtung des Netzes 8n8s auf. Im Vergleich zu dem Netz 8n1s erhöht sich die Latenz um 49.3%. Die Durchführung der Tests mit 16 signierenden Knoten ist mit der verwendeten Testumgebung nicht für alle Testdatengrößen möglich, da die Systemressourcen nicht ausreichen. Die Performance-Tests werden dennoch soweit wie möglich auch mit diesem Netz ausgeführt. Die untere Grenze des Skalierungsverhaltens bildet das Netz 1n1s. Aus den Datenpunkten 1 KB und 1 MB kann der Verlauf annährend mit der Funktion $f(x) = 0.0071x + 7.8328$ abgebil-

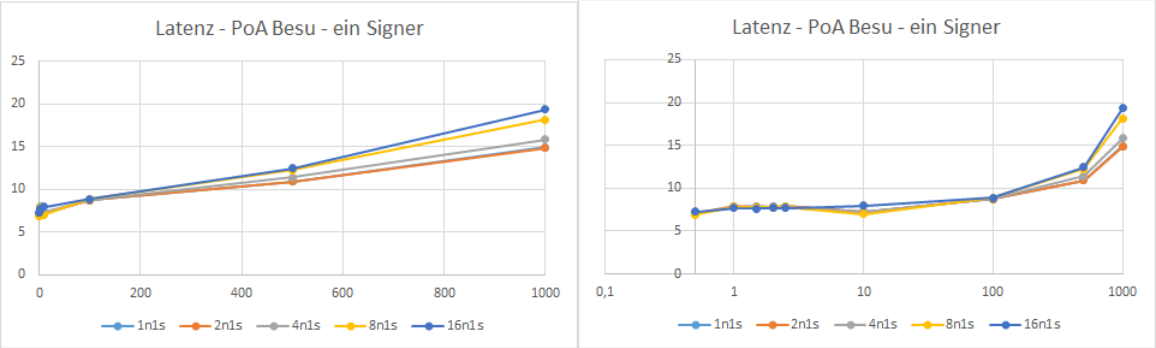

Abbildung 6.1: Latenz: Besu-Netze (PoA) mit einem signierendem Knoten

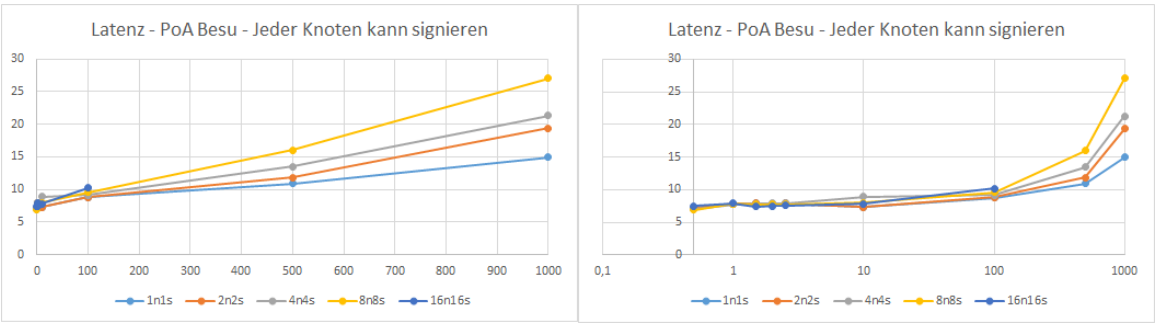

Abbildung 6.2: Latenz: Besu-Netze (PoA) - Jeder Knoten signiert

det werden. Der größte Einfluss durch die Skalierung der signierenden Knoten kann durch das Netz 8n8s beobachtet und mittels der Funktion $f(x) = 0.019269x + 7.80073$ beschrieben werden. Diese wird wie zuvor aus den Datenpunkten für 1 KB und 1 MB berechnet. Um das Skalierungsverhalten weiter zu analysieren, werden für alle Netze die Skalierungsfunktionen gebildet. Eine Übersicht der Funktionen ist in Tabelle 4 im Anhang zu finden. Mit Ausnahme der Funktion für das Netz 16n16s, wurden alle Funktionen aus den Datenpunkten für 1 KB und 1 MB abgeleitet. Die Funktion zur Beschreibung des Netzes 16n16s wird zwecks Annahmen aus den Datenpunkten für 1 KB und 10 KB gebildet. In Abbildung 6.3 werden die Steigungen (m) der Skalierungsfunktionen im Zusammenhang mit der Anzahl der Knoten in einem Diagramm aufgetragen. Dabei wird wie zuvor zwischen Netzen mit einem und mehreren signierenden Knoten unterschieden. Aus der Abbildung geht hervor, dass die Latenz in Netzen mit nur einem signierenden Knoten, nur geringfügig von der Anzahl der Knoten abhängt, da sich die Steigung der Funktionen kaum verändert. Dies bestätigt somit die zuvor getätigte Einschätzung. Auch die Beobachtung bezüglich der Netze in denen die Anzahl signierender Knoten zunimmt, wird durch den Graphen untermauert. In dem Zusammenhang ist besonders der potenzielle Verlauf für das Netz 16n16s interessant, welcher durch die gestrichelte Linie angedeutet wird. Demnach könnte es sein, dass der Einfluss durch die Anzahl der signierenden Knoten im weiteren Verlauf stagniert. Die von [64, S. 8] beschriebene Beobachtung, die Latenz steige an, wenn die Senderate erhöht wird, stellt sich im Zuge der Performance-Tests dieser Arbeit als zutreffend heraus. Jedoch erhöht sich die Latenz beim Speichern der Provenance-Graphen im Unterschied zur Arbeit [64] stark.

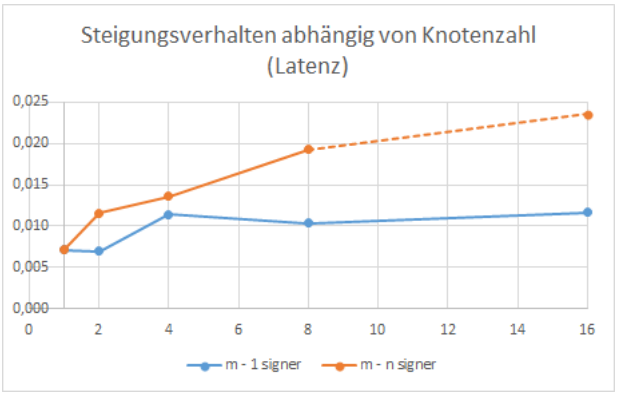

Abbildung 6.3: Latenz: Besu (PoA) - Steigungsverhalten der Skalierungsfunktionen

Durchsatz

Die Messergebnisse des Durchsatzes sind in Tabelle 3 aufgelistet. Sie bestehen aus dem jeweils maximalen Durchsatzwert in *tps*, sowie der Senderate mit welcher dieser erreicht wurde. Zusätzlich ist in den Reports die durchschnittliche Latenz zu den Einträgen erfasst.
Abbildung 6.4 zeigt die Messwerte bis zu einer Testdatengröße von 10 KB für Netze mit einem signierenden Knoten. Der Durchsatz sowie die dazugehörige Senderate erinnern an eine Logarithmus-Funktion. Die ermittelten Werte fallen für die Testdatengrößen zwischen 0.5 und 2.5 stark ab. Ab der Testdatengröße von 10 KB ist der Durchsatz bereits relativ gering und stagniert anschließend beinahe, bis für die Testdatengröße von 1 MB der Wert von 0.1 tps erreicht ist. Diese Beobachtung gilt für jedes Netz. Das erste Intervall zwischen 0.5 KB und 2.5 KB lässt sich annährend durch eine quadratische Funktion beschreiben. Der restliche Verlauf lässt sich über zwei Geraden abbilden. Das Intervall für Testdatengrößen zwischen 100 KB und 1 MB erscheint für Untersuchungen jedoch nicht interessant, da eine Eingrenzung auf den bereits kleinen Wertebereich zwischen 0.1 und 0.9 möglich ist.
Der höchste Durchsatz kann im Rahmen der Untersuchung des Netzes 1n1s beobachtet werden. Für Werte bis 2.5 KB können diese in Annährung durch die Funktion $f(x) = 8.133x^2 - 40x + 63.967$ nachgebildet werden. Die niedrigsten Durchsatzwerte erreicht das Netz 8n1s. Das Skalierungsverhalten kann dafür ebenfalls nur in Annährung durch die Funktion $f(x) = 9.1x^2 - 41.1x + 53.875$ beschrieben werden. Zusätzlich kann beobachtet werden, dass der Verlauf der max. Senderaten im Allgemeinen Ähnlichkeiten mit dem Verlauf des max. Durchsatzes aufweist.
Die erzielten Messergebnisse für Netze in denen alle Knoten signieren können, werden in Abbildung 6.5 betrachtet. Der maximale Durchsatz verhält sich hinsichtlich des Verlaufes ähnlich zu dem der Netzen mit einem signierenden Knoten. Die Senderaten über welche die Werte erzielt werden, ist für kleine Transaktion mit der Größe von 0.5 KB halbiert. Ab einer Testdatengröße von 1 KB gleichen sich die Werte jedoch wieder an. Die niedrigsten Ergebnisse werden im Zuge der durchgeführten Tests erneut von dem Netz mit den meisten Knoten erzielt. Der Verlauf des maximalen Durchsatze für das Netz 8n8s kann durch die Funktion $f(x) = 2.9x^2 - 15.25x + 25.6$ für Testgrößen bis 2.5 KB beschrieben werden. Die Größe der Transaktionen hat somit einen großen Einfluss auf den maximalen Durchsatz, welcher beim Anstieg den Einfluss durch die Anzahl der Knoten reduziert. Maftei et al. leiten in Ihrer Ausarbeitung [49] ab, dass sich der Durchsatz verringern kann, wenn die Blöcke voll sind und keine weiteren Transaktionen aufnehmen können. Dies kann in diesem Fall als Ursache ausgeschlossen werden, da das konfigurierte Gas Limit von $9 \cdot 10^{15}$ theoretisch deutlich mehr

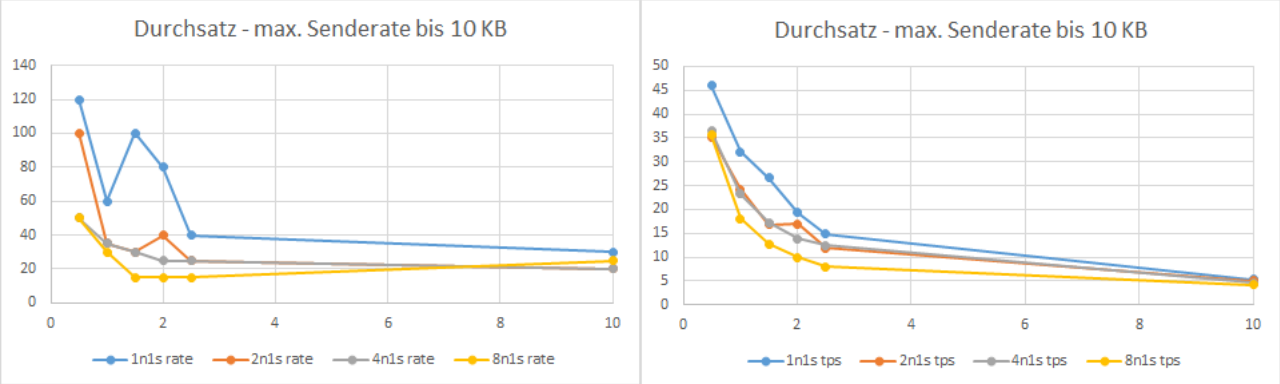

Abbildung 6.4: Besu (PoA) - Durchsatz bis 10 KB, ein Signer

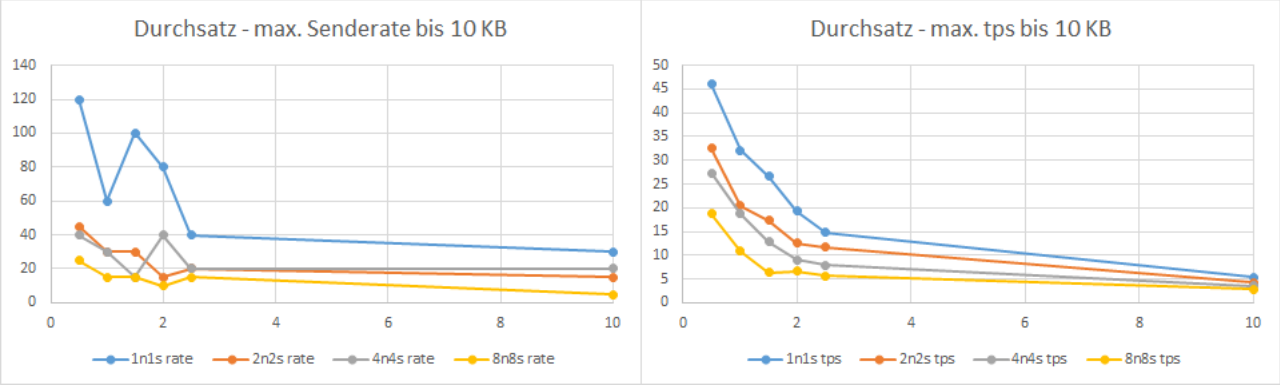

Abbildung 6.5: Besu (PoA) - Durchsatz bis 10 KB, alle Knoten signieren

Transaktionen mit der Größe von 1 MB aufnehmen können sollte.
Aus den erzeugten Reports geht hervor, dass die Latenz mit der Erhöhung der Senderate ansteigt. Als Beispiel ist der Report „report_besu_poa_2n_1s_ohne_05_w2.html“ zu nennen. Zusätzlich kann beobachtet werden, dass der Durchsatz durch die Erhöhung der Senderate bis zu einem gewissen Punkt gesteigert werden kann. Die Beobachtung wird auch in der Arbeiten [64] beschrieben. Zudem stellen die Autoren fest, dass der Anstieg synchron stattfindet. Aus den Testergebnissen der vorliegenden Arbeit geht in der Hinsicht kein 1:1 Verhältis zwischen der Erhöhung der Senderate und dem gemessenen Durchsatz hervor. Zudem ist erkennbar, dass sich der Anstieg des Durchsatzes mit dem Anstieg der Testdatengröße reduziert.

Gas Limit

Die Durchführung der zuvor definierten Workloads ist mit einem Gas Limit von $9 \cdot 10^9$ nicht sinnvoll. Die in den oberen Abschnitten analysierten Tests wurden zum Vergleich mit einem Gas Limit von $9 \cdot 10^{15}$ ausgeführt. Die Performance-Tests mit den Workloads *w1-1* und *w1-3* laufen zwar durch, sind aber für das niedrigen Gas Limit mit zu schnellen Senderaten definiert und zu vielen Transaktionen definiert. Obwohl genug Platz in innerhalb der Blöcke sein sollte, werden pro Block nur 1 - 3 Transaktionen gespeichert. Dies reduziert den Durchsatz auf 0.1 für alle Testdatengrößen. Somit wird der Durchsatz deutlich gesenkt und ein starker Anstieg der Latenz ist zu beobachten.
Bei anschließenden Tests mit einem Gas Limit von $9 \cdot 10^{10}$ sind die Latenz-Test wieder normal

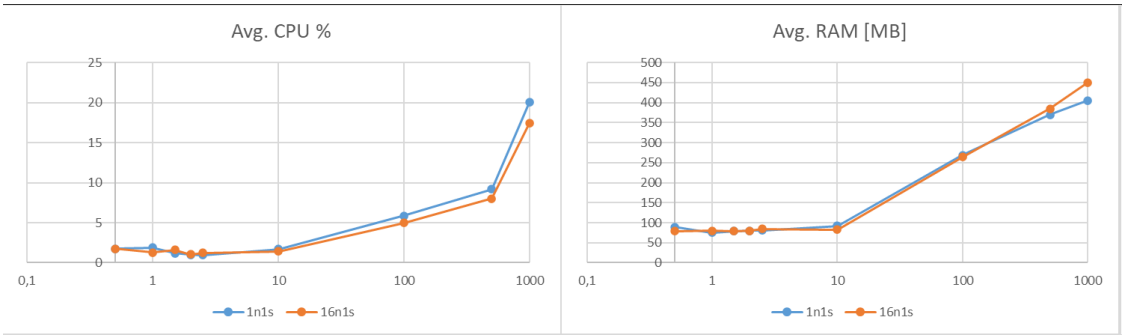

Abbildung 6.6: Vergleich des druchschnittlichen Ressourcenbedarfs bei einem Signer

durchführbar und erreichen ähnliche Ergebnisse wie beim maximalen Gas Limit.

CPU und RAM

Zur Untersuchung des Ressourcenbedarfs, wird die durchschnittliche CPU und RAM Auslastung aller Knoten erfasst. Eine Messung umfasst den Zeitraum der Ausführung eines Testsets mit einer Testdatengröße. Die gemessenen Auslastungen basieren auf den Tests von Tabelle 1. Die Abbildung 6.6 stellt die Skalierung der Ressourcen ohne Last für die Netze 1n1s und 16n1s dar. Diese wurden gewählt, da sie die größtmögliche Differenz zwischen der genutzten Anzahl an Knoten bilden. Erkennbar ist, dass der Bedarf an Ressourcen pro Knoten für beide Netze über den gesamten Testzeitraum relativ gleich bleib. Anzumerken ist jedoch, dass der Gesamtbedarf des Netze 16n1s etwa um das 16-fache erhöht ist, als der des Netzes 1n1s.
Interessant ist der unterschiedliche Verlauf der Graphen. Für die Tests bis 10 KB verlaufen beide Graphen linear. Für die Testdatengröße von 100 KB erhöht sich die Steigung der CPU Auslastung deutlich und auch der RAM-Bedarf steigt sprunghaft an. Ab 500 KB steigt die CPU Auslastung weiterhin linear an, während der RAM-Bedarf abflacht. Insgesamt kann der RAM-Bedarf für Netze mit einem Knoten über eine logarithmische Funktion beschrieben werden. Eine Annährung kann über die Funktion $f(x) = 123.667 \cdot log(x) + 79$ angegeben werden. Die CPU Auslastung hat einen relativ linearen Charakter und kann über die Funktion $f(x) = 0.01819x + 1.8518$ beschrieben werden. Aus den Ergebnissen geht hervor, dass der Bedarf an Ressourcen linear mit der Anzahl der Knoten steigt. Zudem kann beobachtet werden, dass die Größe der Transaktionen einen signifikanten Einfluss auf den Ressourcenbedarf hat. Die CPU Auslastung steigt linear in Abhängigkeit zur Transaktionsgröße an. Der RAM-Bedarf steigt in Abhängigkeit zur Transaktionsgröße bis 100 KB sehr stark an. Anschließend reduziert sich der Einfluss auf den RAM-Bedarf mit zunehmender Transaktionsgröße wieder. Der Ressourcenbedarf bei mehreren signierende Knoten wird in Abbildung 6.7 verdeutlicht. In dieser werden alle Netze abgebildet, da sich zum Teil große Unterschiede zwischen diesen ergeben. Aus dem Diagramm geht hervor, dass die signierenden Knoten wie ein Faktor auf den Ressourcenbedarf wirken. Eine Ausnahme davon bildet das Netz 2n2s. Dieser Umstand verdeutlicht sich ganz besonders bei Transaktionen überhalb der Größe von 100 KB. Abbildung 6.8 visualisiert die Skalierung des Ressourcenbedarfs durch die Anzahl der signierenden Knoten für die CPU Auslastung. Die Anzahl der Signer wirkt entsprechend der ersten Schätzung als Faktor, was im Zusammenspiel mit Transaktionen über der Größe von 100 KB an Relevanz gewinnt. Aus dem Graphen für die CPU Auslastung in Abbildung 6.7 ist zu entnehmen, dass die Testumgebung beim Speichern eines Provenance-Graphen von 1 MB nahezu voll ausgelastet ist. Der Einfluss der Anzahl der Signer auf den RAM-Bedarf ist nicht ganz

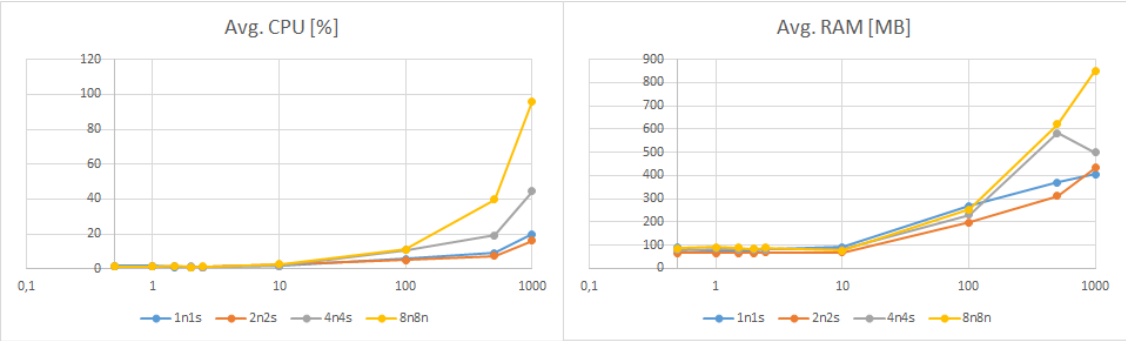

Abbildung 6.7: Vergleich des druchschnittlichen Ressourcenbedarfs bei mit n Signern

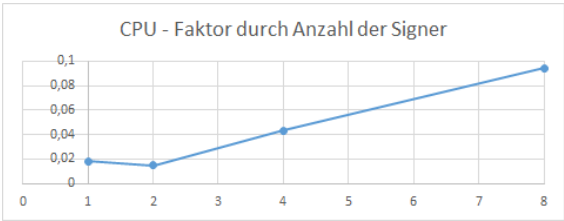

Abbildung 6.8: Faktor durch Signer auf Ressourcenbedarf

klar. Auffällig ist der letzte Datenpunkt des Netzes 4n4s. Dieser wirkt wie ein Ausreißer, da die Messungen sonst sehr ähnlich zu denen des Netzes 8n8s sind. Es könnte sich auch bei der Messung mit der Testdatengröße von 500 KB um eine Abweichung handeln. Weiterhin deutet der Verlauf der Graphen für die Netze 4n4s und 8n8s weniger auf eine logarithmische Funktion hin, als auf eine quadratische. So liefert die Funktion $f(x) = -0.001x^2 + 1.747x + 89.154$ eine bessere Annährung an für erfassten Werte des Netzes 8n8s, als der Logarithmus.
Alle verwendeten Daten sind im Anhang in den Tabellen 5 und 6 zu finden.
Aus der in den Grundlagen beschrieben Abstraktionsschicht in Abschnitt 3.2.6 geht hervor, dass der Execution Layer Systemressourcen wie die CPU und den RAM benötigt. Der Anstieg des Ressourcenbedarfs bei Transaktionen über 100 KB könnte darauf hindeuten, dass besonders der Execution Layer von der Ausführung großer Transaktionen belastet wird. Im Zuge dessen könnte der Anstieg des Ressourcenbedarfs auf die Ausführung des Smart Contracts zurückzuführen sein.

6.1.2 Grenzen des Ethereum Mainnets und von Bloxberg

Der Workload *w4* wird zur Bestimmung des Gas-Bedarfs aller erzeugten Testdaten eingesetzt. Über das Monitoring Tool Chainlens, können die Blöcke der Blockchain anschließend untersucht werden. Dieses wird als Docker Container über den Befehl „NODE_ENDPOINT=http://172.16.239.1:8545 PORT=26000 docker compose up“ gestartet und über den HTTP Endpunkt mit einem Besu-Client verbunden. Über eine bereitgestellte Webseite, können dann die Blöcke betrachtet werden. So kann auch das genutzte Gas für jeden Block eingesehen werden. Der Gas-Bedarf aller Testdaten ist in Tabelle 6.2 zu finden. Das Ergebnis wird stichprobenartig mit dem Testdatensatz für 10 KB in einem lokalen Geth-Netz überprüft. Dieses wird auch genutzt, um die berechneten Ergebnisse zu bestätigen. Verwendet wird hierzu der Geth-Client in der Version 1.13.15. Da es sich bei Bloxberg gemäß der Dokumentation [5] ebenfalls um eine Implementierung von Ethereum handelt, Das

Graphgröße	Genutztes Gas	Blockgröße
0,5	485.113	1,37
1	894.039	1,94
1,5	1.302.558	2.518
2	1.620.806	2.966
2,5	1.916.018	3.382
10	7.457.908	11.190
100	71.842.840	101.883
500	364.310.214	513.275
1000	743.591.383	1.045.435

Tabelle 6.2: Gas-Bedarf pro Testdatengröße

```
twigm@Ubuntu-2204-jammy-amd64-base:~/performance_tests/scripts$ python test_contract.py
tx hash: 0x9205cbcd8747b3e9a58df65dba15e50e883e7ed5d53ee9720d42034d379aafa3
tx info:
--------
status: 1
block number: 109
transactionHash:0x9205cbcd8747b3e9a58df65dba15e50e883e7ed5d53ee9720d42034d379aafa3
cumulativeGasUsed: 7457908
gasUsed: 7457908
```

Abbildung 6.9: Kontrolle des Gas-Bedarfs mit Geth-Client

Netz wird dafür mit einem Knoten und einem „gasLimit“ von 30.000.000 gestartet. Sowohl die Genesis-Datei als auch das Start-Skript sind im Anhang zu finden. Über das Python-Skript „deploy_contract.py“ wird der Smart Contract zunächst erzeugt und mit dem Skript „test_contract.py“ der Testdatensatz mit 10 KB eingespielt. Mit der Ausgabe in Abbildung 6.9 wird die einheitliche Berechnung des Gas-Bedarfs zwischen Besu und Geth somit bestätigt.

Hinsichtlich der Forschungsfrage *RQ3* soll die Datenmenge bestimmt werden, welche im Ethereum Mainnet und der Blockchain Bloxberg maximal gespeichert werden können. Sowohl das Ethereum Mainnet, als auch Bloxberg bieten zur Betrachtung der Blockchain einen Blockexplorer an. Über Etherscan kann das Mainnet unter der URL https://etherscan.io/ eingesehen werden. Der Blockexplorer von Bloxberg ist über die URL https://blockexplorer.bloxberg.org/ zu erreichen. Nach der Selektion eines Blocks können Teile der Konfiguration des Netzes eingesehen werden. So ist Ethereum ist mit einem Gas Limit von $30 \cdot 10^6$ und Bloxberg mit einem Gas Limit von $10 \cdot 10^6$ konfiguriert. Aus diesen Angaben können in Verbindung mit den Ergebnissen des Workloads *w4* Schätzungen vorgenommen werden.
Am nächsten liegt der Gas-Bedarf des Testdatensatzes mit 10 KB, welcher $7.5 \cdot 10^6$ Gas belegt. Darüber liegt der Testdatensatz mit 100 KB und einem Gas-Bedarf von $71.8 \cdot 10^6$. Aus den Ergebnissen in Tabelle 6.2 geht ein linearer Anstieg hervor. Aus den Punkten für die Datensätze mit 10 KB und 100 KB, wird die Funktion $f(x) = 715388.13x + 304026.66$ zur Bestimmung des genutzten Gas für eine Datenmenge gebildet. In diese werden die Gas-Limits von Ethereum und Bloxberg eingesetzt und die Gleichung nach x aufgelöst. Somit ergibt sich für Ethereum eine theoretische maximale Datenmenge von 41.5 KB und für Bloxberg eine maximale Datenmenge von 13.5 KB.

Die berechneten Ergebnisse werden überprüft, indem Provenance-Graphen erzeugt werden, welche im Dateiformat den Datengrößen von 13 KB, 14 KB, 15 KB, 41 KB, 42, 43 KB und 44 KB entsprechen. Im Anschluss werden diese der Reihe nach mit dem Skript „test_contract.py“ eingespielt. Die Konsolenausgabe bestätigt die Schätzungen. Nachdem das Einspielen Für den Provenance-Graphen mit 41 KB noch funktioniert und 29718615 Gas benötigt, führt gleiche Test mit 42 KB zu der folgenden Fehlermeldung: „ValueError: Contract does not appear to be deployable within the current network gas limits. Estimated: 30688666. Current gas limit: 30000000“. Ähnlich verhält es sich mit den Provenance-Graphen mit der Dateigröße von 13 KB. Das Einspielen ist erfolgreich und benötigt 9751934 Gas. Die Testdaten mit der Dateigröße von 14 KB überschreiten die Zielgröße von 10.000.000 mit einem Gas-Bedarf von 10319898. Eine weiter Grenze existiert unabhängig vom Gas Limit für den Ethereum Client. So

Kapitel 7

Fazit und Ausblick

Die Ziele der Arbeit konnten nicht vollständig erreicht werden. Die in den Forschungsfragen *RQ1* und *RQ2* beschriebenen Untersuchungen eine PoS Netzes konnten nicht durchgeführt werden. Die Tests wurden zwar teilweise durchgeführt, konnten jedoch nicht mehr im Rahmen der Arbeit ausgewertet werden. Einen Grund stellt der Workload *w1* dar, welcher für alle Netze ausgeführt wurde und auf dessen Ergebnisse Teile der Auswertung basierten. Wie in der Evaluation erläutert, weisen die erzielten Ergebnisse des Workloads insbesondere im Zusammenhang mit Testdaten ab 100 KB größere Abweichungen auf. Diese kommen durch zeitgleich offene Transaktionen im Transaktionspool der Blockchain zustande, welche mit ansteigender Transaktionsgröße einen ansteigenden Einfluss auf die Latenz haben und diese erhöhen. Der Effekt konnte im Zuge der Evaluation experimentell festgestellt werden. Zum Zeitpunkt der Erstellung des Workloads, war der beschrieben Effekt nicht bekannt, da im Rahmen der Literaturrecherche keine Arbeiten identifiziert werden konnten, welche sich mit größeren Transaktionsgrößen beschäftigt haben. Aus diesem Grund wurde der Workload *w1* auf Basis der Arbeiten [35], [63] und [64] erstellt, welche sich ebenfalls mit Performance-Tests und Hyperledger Caliper beschäftigen. Da diese keine großen Transaktionen ausführen, führen die in den genannten Arbeiten definierten Performance-Tests nicht zu Problemen. Nach Feststellung der Abweichungen wurden neue Performance-Tests definiert und für alle Testnetze erneut durchgeführt. So ersetzen die Workloads *w1-1* und *w1-3* zusammen den Workload *w1*, sind dafür aber zeitaufwendiger und die Ergebnisse basieren auf einer kleineren Testbasis.
Auf Grundlage der neuen Ergebnisse wurden die Skalierungsfaktoren im Hinblick der Forschungsfragen *RQ1* und *RQ2* für PoA-Netze untersucht. Bezüglich der Skalierung wurden unterschiedliche Parameter hinsichtlich der Latenz, des Durchsatzes, der durchschnittlichen CPU Auslastung und des RAM-Bedarfs betrachtet. Bei den genannten Bereichen handelt es sich um die Metriken, welche im Zuge der Forschungsfrage *RQ4* identifiziert wurden. Zusätzlich zu diesen wurden als Methode die Durchführung von Benchmarks gewählt, welche mit Hyperledger Caliper durchgeführt wurden. Als weitere Methode wurden zudem experimentelle Tests gewählt, um die Forschungsfrage *RQ3* zu beantworten. Um die Durchführung zu ermöglichen, wurde eine Testumgebung aufgesetzt und ein Verfahren zum Speichern von Provenance-Graphen im Hinblick auf *RQ6* ausgewählt. Zudem wurden zur Beantwortung von *RQ5* Möglichkeiten und Ansätze recherchiert, um synthetische Provenance-Graphen zu generieren, damit diese für die Performance-Tests verwenden werden können. Ein Graphgenerator konnte prototypisch umgesetzt und Provenance-Graphen entsprechend vorgegebener Größen erzeugt werden. Nach der Durchführung der Performance-Tests wurden die erfassten Metriken hinsichtlich ihrer Skalierung durch die Parameter Netzgröße, Anzahl signierender Knoten und Testdatengröße ausgewertet.

Für die Metrik *Latenz* konnte ein erhöhender Einfluss durch die Größe des Netzes festgestellt werden. Die Latenz skaliert für Netze mit einem signierenden Knoten linear in Abhängigkeit der Transaktionsgröße und der Anzahl der Knoten. Demnach müssen die Skalierungsfunktionen für jede Netzgröße individuell erstellt werden. Anschließend kann über diese eine Annährung an die Latenz für eine beliebige Transaktionsgröße berechnet werden. Im Rahmen der Evaluation wurde eine solche Skalierungsfunktion für das kleinste und größte getestete Netz erstellt.
Die gleiche Auswertung wurde für Netze durchgeführt in denen alle Knoten signieren können. Auch für diese wurde eine lineare Skalierung der Latenz in Abhängigkeit von der Anzahl der Knoten und der Transaktionsgröße festgestellt. Um den Einfluss durch die Anzahl der Knoten genauer zu untersuchen, wurden die Skalierungsfunktionen für alle getesten Netze berechnet und bezüglich ihres Steigungsverhaltens ausgewertet. Aus der Auswertung geht hervor, dass die Latenz in Netzen mit nur einem signierenden Knoten nur gering von der Anzahl der Knoten abhängt. In Netzen in denen alle Knoten signieren können, hat die Anzahl der Knoten einen stärkeren und linear ansteigenden Einfluss.
Für die Metrik *Durchsatz* wurden erneut Netze untersucht in denen ein Knoten oder alle Knoten signieren können. In beiden Fällen konnte für alle getesten Netze festgestellt werden, dass der Durchsatz für Transaktionen bis 2.5 KB quadratisch abnimmt und sich anschließend eher linear skaliert. Da die Skalierungsfunktionen erneut individuell für die Netze erstellt werden müssen, da die Anzahl der Knoten ebenfalls einen Einfluss hat, welcher jedoch mit der Zunahme der Transaktionsgröße abnimmt, wurden die Funktionen für das Netz mit dem höchsten und dem niedrigsten Durchsatz berechnet. Für die Metrik *CPU Auslastung* konnte für Netze mit einem signierende Knoten festgestellt werden, dass das die summierte CPU Auslastung linear mit der Anzahl der Knoten skaliert, da alle Knoten eine sehr ähnliche Auslastung haben. Auch die Größe der Transaktionen hat einen linear skalierenden Einfluss auf die CPU Auslastung. In dem Fall konnte ein Skalierungsfunktion berechnet werden, welche für alle getesten Netze mit einem signierenden Knoten anwendbar ist. Werden Netze betrachtet in denen alle Knoten signieren können fungiert die Anzahl der signierenden Knoten als Skalierungsfaktor. Nur das Netz mit zwei Knoten bildet hiervon eine Ausnahme.
Für die Metrik *RAM* wurde ebenso, wie für die CPU festgestellt, dass der Gesamtbedarf für Netze mit einem signierenden Knoten linear ansteigt. In Abhängigkeit von der Transaktionsgröße skaliert der Bedarf logarithmisch und es konnte eine Funktion aufgestellt werden, welche den RAM-Bedarf annährend für Netze unterschiedlicher Größe beschreibt. Für Netze mit mehreren signierenden Knoten, konnte die Skalierung für das Netz mit dem höchsten Bedarf über eine exponentielle Annährung beschrieben beschrieben werden. Zusammen mit der logarithmischen Funktion für das Netz mit dem kleinsten Bedarf konnte somit Korridor angegeben werden, in welchem sich die Werte der anderen Netze befinden. Dies hat den Grund, dass für ein Netz die Skalierung aufgrund eines vermutlichen Ausreißers nicht sicher bestimmt werden konnte.
Die somit festgestellten Skalierungsfaktoren und Funktionen beantworten die Forschungsfragen *RQ1* und im Hinblick auf die Netzgrößen auch *RQ2* der Arbeit. Die Betrachtung des Gas Limits stellt den zweiten Teil der Forschungsfrage dar. In einem Experiment konnte festgestellt werden, dass die alleinige Senkung des Gas Limits auf $9 \cdot 10^9$ den Durchsatz für alle Testdatengrößen auf 0.1 tps senkt. Zudem konnte ein starker Anstieg der Latenz beobachtet werden. Bezüglich der Forschungsfrage *RQ3* konnte zudem über die Anpassung des Gas Limits festgestellt werden, dass auf dem Etherem Mainnet maximal Transaktionen von 41 KB ausgeführt werden können und bei Bloxberg die Grenze bei 13 KB liegt.
Im Zuge der Auswahl des genutzten Ethereum-Clients wurden festgestellt, dass der Geth-

Client nur Transaktionen bis 128 KB ausführen kann. Für Besu liegt die Grenze aufgrund der Begrenzung eines Buffers bei 4 MB. Interessant wäre daher die Betrachtung anderer Distributed Ledger Technologien. Beispielsweise werden in [?] auch Systeme speziell zum Speichern von gerichteten azyklischen Graphen betrachtet. So gibt es beispielsweise ein System namens „ProvChain“, welches zum Speichern von Provenance-Graphen konzipiert ist.
Weiterhin ist die Untersuchung eines PoS-Netzes im Rahmen der Arbeit offen geblieben und die Auswertung von Lese-Operationen wurde auch ebenfalls nicht systematisch untersucht. Zudem gilt der Konsensalgorithmus Clique seit Geth v1.14 als veraltet. Daher wären Performance-Tests mit großen Transaktionen für weitere Konsensalgorithmen wie QBFT oder IBFT interessant. Außerdem wurde das Speichern mehrerer Daten parallel und über mehrere Ethereum-Clients nicht untersucht. Eine weitere Möglichkeit stellt die genauere Untersuchung der horizontalen Skalierung dar. Die Arbeit hat sich ausschließlich mit Knoten auf dem gleichen Server beschäftigt. Weiterhin könnten Performance-Tests auf physikalisch getrennten Knoten durchgeführt werden.
Grundsätzlich bietet auch der implementierte Graph-Generators Verbessungspotenzial indem die Verteilung der Knoten geschickter durchgeführt oder gezielt die Struktur des Graphen gesteuert werden kann, jedoch ist das Anwendungsszenario in anderen Kontexten vermutlich zu spezifisch, da es andere Generatoren gibt, welche realistischere Daten erzeugen.
PoS Betrachtung vernachlässigt Fokus auf PoA mit Besu, da dieses aufgrund der Akzeptanz größerer Transaktionen interessanter für die Tests ausfällt.

Literaturverzeichnis

[1] *Command line options*. https://besu.hyperledger.org/public-networks/reference/cli/options, . – Zuletzt besucht: 01.06.24

[2] *Configure Clique consensus*. https://besu.hyperledger.org/development/private-networks/how-to/configure/consensus/clique, . – Zuletzt besucht: 01.06.24

[3] *Configure free gas networks*. https://besu.hyperledger.org/23.4.0/private-networks/how-to/configure/free-gas, . – Zuletzt besucht: 01.06.24

[4] *Create a private network using Clique*. https://besu.hyperledger.org/development/private-networks/tutorials/clique, . – Zuletzt besucht: 01.06.24

[5] *Documentation*. https://bloxberg.org/developers-hut/#documentation, . – Zuletzt besucht: 06.06.24

[6] *Hyperledger Besu Ethereum client*. https://besu.hyperledger.org/, . – Zuletzt besucht: 23.11.23

[7] *Hyperledger Caliper*. https://hyperledger.github.io/caliper/, . – Zuletzt besucht: 23.11.23

[8] *Hyperledger Caliper - Getting Started*. https://hyperledger.github.io/caliper/v0.5.0/getting-started/, . – Zuletzt besucht: 02.06.24

[9] *Network ID and chain ID*. https://besu.hyperledger.org/development/public-networks/concepts/network-and-chain-id, . – Zuletzt besucht: 01.06.24

[10] *Next-Gen Corda 5.2*. https://docs.r3.com/en/platform/corda/5.2.html, . – Zuletzt besucht: 07.06.24

[11] *Private network command line options*. https://besu.hyperledger.org/development/private-networks/reference/cli/options, . – Zuletzt besucht: 01.06.24

[12] *ProvValidator*. https://openprovenance.org/service/validator.html, . – Zuletzt besucht: 05.06.24

[13] *Static nodes*. https://besu.hyperledger.org/development/public-networks/how-to/connect/static-nodes#static-nodesjson-file, . – Zuletzt besucht: 01.06.24

[14] *Enterprise Ethereum Architecture Stack*. https://entethalliance.org/wp-content/uploads/2020/12/EEA_Architecture_Stack.pdf, 2020. – [Letzter Zugriff 18.12.2023]

[15] Abhishek, P. ; Narayan, D. ; Altaf, H. ; Somashekar, P. : Performance Evaluation of Ethereum and Hyperledger Fabric Blockchain Platforms. In: *2022 13th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. IEEE. – ISBN 978–1–66545–262–5, 1–5

[16] Acharya, V. ; Eswararao Yerrapati, A. ; Prakash, N. : *Oracle Blockchain Quick Start Guide: a Practical Approach to Implementing Blockchain in Your Enterprise*. Birmingham : Packt Publishing, Limited, 2019. – ISBN 978–1–78980–130–9. – OCLC: 1119637273

[17] al., C. S.: *Nodes and clients*. https://ethereum.org/en/developers/docs/consensus-mechanisms/poa/, 2023. – [Letzter Zugriff 7.6.2024]

[18] al., C. S.: *Nodes and clients*. https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/, 2023. – [Letzter Zugriff 7.6.2024]

[19] al., C. S.: *Nodes and clients*. https://vitalik.eth.limo/general/2020/11/06/pos2020.html, 2023. – [Letzter Zugriff 7.6.2024]

[20] al., C. S.: *Nodes and clients*. https://academy.binance.com/en/articles/proof-of-authority-explained, 2023. – [Letzter Zugriff 7.6.2024]

[21] al., C. S.: *Nodes and clients*. https://ethereum.org/de/developers/docs/ethereum-stack/, 2023. – [Letzter Zugriff 7.6.2024]

[22] al., C. S.: *Nodes and clients*. https://ethereum.org/de/developers/docs/smart-contracts/, 2023. – [Letzter Zugriff 7.6.2024]

[23] al., C. S.: *Nodes and clients*. https://www.coinbase.com/de/learn/crypto-basics/what-is-a-smart-contract, 2023. – [Letzter Zugriff 7.6.2024]

[24] al., C. S.: *Nodes and clients*. https://ethereum.org/en/developers/docs/nodes-and-clients/, 2023. – [Letzter Zugriff 16.12.2023]

[25] Campos, P. ; Dahir, N. ; Bonney, C. ; Trefzer, M. ; Tyrrell, A. ; Tempesti, G. : XL-STaGe: A cross-layer scalable tool for graph generation, evaluation and implementation. In: *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, IEEE. – ISBN 978–1–5090–3076–7, 354–359

[26] Dabbagh, M. ; Kakavand, M. ; Tahir, M. ; Amphawan, A. : Performance Analysis of Blockchain Platforms: Empirical Evaluation of Hyperledger Fabric and Ethereum. In: *2020 IEEE 2nd International Conference on Artificial Intelligence in Engineering and Technology (IICAIET)*, IEEE. – ISBN 978–1–72816–946–0, 1–6

[27] Dang, T. K. ; Anh, T. D.: A Pragmatic Blockchain Based Solution for Managing Provenance and Characteristics in the Open Data Context. Version: 2020. http://dx.doi.org/10.1007/978-3-030-63924-2_13. In: Dang, T. K. (Hrsg.) ; Küng, J. (Hrsg.) ; Takizawa, M. (Hrsg.) ; Chung, T. M. (Hrsg.): *Future Data and Security*

Engineering Bd. 12466. Springer International Publishing. – DOI 10.1007/978–3–030–63924–2$_1$3. – – $ISBN978 - -3 - -030 - -63923 - -5978 - -3 - -030 - -63924 - -2, 221 - -242. - -SeriesTitle : LectureNotesinComputerScience$

[28] Demichev, A. ; Kryukov, A. ; Prikhod'ko, N. : Business Process Engineering for Data Storing and Processing in a Collaborative Distributed Environment Based on Provenance Metadata, Smart Contracts and Blockchain Technology. In: *Journal of Grid Computing* 19 (2021), März, Nr. 1, 3. http://dx.doi.org/10.1007/s10723-021-09544-4. – DOI 10.1007/s10723–021–09544–4. – ISSN 1570–7873, 1572–9184

[29] Dinh, T. T. A. ; Wang, J. ; Chen, G. ; Liu, R. ; Ooi, B. C. ; Tan, K.-L. : BLOCKBENCH: A Framework for Analyzing Private Blockchains. In: *Proceedings of the 2017 ACM International Conference on Management of Data*, ACM. – ISBN 978–1–4503–4197–4, 1085–1100

[30] Dong, Z. ; Zheng, E. ; Choon, Y. ; Zomaya, A. Y.: DAGBENCH: A Performance Evaluation Framework for DAG Distributed Ledgers. In: *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE. – ISBN 978–1–72812–705–7, 264–271

[31] Dreyer, J. ; Fischer, M. ; Tönjes, R. : Performance analysis of hyperledger fabric 2.0 blockchain platform. In: *Proceedings of the Workshop on Cloud Continuum Services for Smart IoT Systems*. ACM. – ISBN 978–1–4503–8131–4, 32–38

[32] El Emam, K. ; Mosquera, L. ; Hoptroff, R. : *Practical synthetic data generation: balancing privacy and the broad availability of data*. First Edition. O'Reilly Media, Inc. – ISBN 978–1–4920–7274–4. – OCLC: on1164815296

[33] Fadhel, N. ; Lombardi, F. ; Aniello, L. ; Margheri, A. ; Sassone, V. : Towards a semantic modelling for threat analysis of IoT applications: a case study on transactive energy. In: *Living in the Internet of Things (IoT 2019)*. Institution of Engineering and Technology. – ISBN 978–1–83953–089–0, 22 (6 pp.)–22 (6 pp.)

[34] Fan, C. ; Ghaemi, S. ; Khazaei, H. ; Musilek, P. : Performance Evaluation of Blockchain Systems: A Systematic Survey. 8, 126927–126950. http://dx.doi.org/10.1109/ACCESS.2020.3006078. – DOI 10.1109/ACCESS.2020.3006078. – ISSN 2169–3536

[35] Fan, C. ; Lin, C. ; Khazaei, H. ; Musilek, P. : Performance Analysis of Hyperledger Besu in Private Blockchain. In: *2022 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*. IEEE. – ISBN 978–1–66549–172–3, 64–73

[36] Firth, H. ; Missier, P. : ProvGen: Generating Synthetic PROV Graphs with Predictable Structure. http://dx.doi.org/10.1007/978-3-319-16462-5_2. In: Ludäscher, B. (Hrsg.) ; Plale, B. (Hrsg.): *Provenance and Annotation of Data and Processes* Bd. 8628. Springer International Publishing. – DOI 10.1007/978–3–319–16462–5_2. – ISBN 978–3–319–16461–8 978–3–319–16462–5, 16–27. – Series Title: Lecture Notes in Computer Science

[37] Graham Bath ; Rex Black ; Alexander Podelko ; Andrew Pollner ; Randy Rice: *ISTQB Certified Tester- Foundation Level Specialist Syllabus Performance Testing*. https://istqb-main-web-prod.s3.amazonaws.com/media/documents/ISTQB-CT-PT_Syllabus_v1.0_2018.pdf

[38] Groth, P. ; Moreau, L. : *PROV-Overview - An Overview of the PROV Family of Documents*. https://www.w3.org/TR/prov-overview/, April 2013. – [Letzter Zugriff 02.06.2023]

[39] Gupta, A. : Data Provenance. http://dx.doi.org/10.1007/978-1-4614-8265-9_1305. In: Liu, L. (Hrsg.) ; Özsu, M. T. (Hrsg.): *Encyclopedia of Database Systems*. Springer New York. – DOI 10.1007/978–1–4614–8265–9_1305. – ISBN 978–1–4614–8266–6 978–1–4614–8265–9, 812–812

[40] Gürsakal, N. ; Celik, S. ; Birisci, E. : *Synthetic data for deep learning: generate synthetic data for decision making and applications with python and R*. Apress. – ISBN 978–1–4842–8586–2

[41] Ioini, N. E. ; Pahl, C. : Trustworthy Orchestration of Container Based Edge Computing Using Permissioned Blockchain. In: *2018 Fifth International Conference on Internet of Things: Systems, Management and Security*. IEEE. – ISBN 978–1–5386–9585–2, 147–154

[42] Ismail, L. ; Hameed, H. ; AlShamsi, M. ; AlHammadi, M. ; AlDhanhani, N. : Towards a Blockchain Deployment at UAE University: Performance Evaluation and Blockchain Taxonomy. In: *Proceedings of the 2019 International Conference on Blockchain Technology*. ACM. – ISBN 978–1–4503–6268–9, 30–38

[43] Kapoor, M. ; Melton, J. ; Ridenhour, M. ; Moyer, T. ; Krishnan, S. : Flurry: A Fast Framework for Provenance Graph Generation for Representation Learning. In: *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. ACM. – ISBN 978–1–4503–9236–5, 4887–4891

[44] Kitchenham, B. : Guidelines for performing Systematic Literature Reviews in Software Engineering / Keele Univerity and University of Durham. 2007 (2.3). – EBSE Technical Report

[45] Kocadag, S. : *Trusted Provenance with Blockchain - A Blockchain-based Provenance Tracking System for Virtual Aircraft Component Manufacturing*. 2023

[46] Kuzlu, M. ; Pipattanasomporn, M. ; Gurses, L. ; Rahman, S. : Performance Analysis of a Hyperledger Fabric Blockchain Framework: Throughput, Latency and Scalability. In: *2019 IEEE International Conference on Blockchain (Blockchain)*. IEEE. – ISBN 978–1–72814–693–5, 536–540

[47] Lautert, F. ; Pigatto, D. F. ; Gomes, L. : A fog architecture for privacy-preserving data provenance using blockchains. In: *2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE. – ISBN 978–1–72818–086–1, 1–6

[48] Leal, F. ; Chis, A. E. ; González–Vélez, H. : Performance Evaluation of Private Ethereum Networks. 1, Nr. 5, 285. http://dx.doi.org/10.1007/s42979-020-00289-7. – DOI 10.1007/s42979–020–00289–7. – ISSN 2662–995X, 2661–8907

[49] Maftei, A.-A. ; Lavric, A. ; Petrariu, A.-I. ; Popa, V. : Performance Evaluation of Block Size Influence on Blockchain-Enabled IoT Data Storage. In: *2023 15th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*. IEEE. – ISBN 9798350321388, 1–4

[50] Margheri, A. ; Masi, M. ; Miladi, A. ; Sassone, V. ; Rosenzweig, J. : Decentralised provenance for healthcare data. In: *International Journal of Medical Informatics* 141 (2020), Sept., 104197. http://dx.doi.org/10.1016/j.ijmedinf.2020.104197. – DOI 10.1016/j.ijmedinf.2020.104197. – ISSN 13865056

[51] Markovic, M. ; Edwards, P. ; Jacobs, N. : Recording Provenance of Food Delivery Using IoT, Semantics and Business Blockchain Networks. In: *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*. IEEE. – ISBN 978–1–72812–949–5, 116–118

[52] Markovic, M. ; Jacobs, N. ; Dryja, K. ; Edwards, P. ; Strachan, N. J. C.: Integrating Internet of Things, Provenance, and Blockchain to Enhance Trust in Last Mile Food Deliveries. In: *Frontiers in Sustainable Food Systems* 4 (2020), Nov., 563424. http://dx.doi.org/10.3389/fsufs.2020.563424. – DOI 10.3389/fsufs.2020.563424. – ISSN 2571–581X

[53] Missier, P. ; Moreau, L. : *PROV-N: The Provenance Notation*. https://www.w3.org/TR/prov-n/, April 2013. – [Letzter Zugriff 02.06.2023]

[54] Missier, P. ; Moreau, L. ; Belhajjame, K. ; B'Far, R. ; Cheney, J. ; Coppens, S. ; Cresswell, S. ; Gil, Y. ; Groth, P. ; Klyne, G. ; Lebo, T. ; McCusker, J. ; Miles, S. ; Myers, J. ; Sahoo, S. ; Tilmes, C. : *PROV-DM: The PROV Data Model*. https://www.w3.org/TR/prov-dm/, April 2013. – [Letzter Zugriff 02.06.2023]

[55] Monrat, A. A. ; Schelen, O. ; Andersson, K. : Performance Evaluation of Permissioned Blockchain Platforms. In: *2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*. IEEE. – ISBN 978–1–66541–974–1, 1–8

[56] Nasrulin, B. ; De Vos, M. ; Ishmaev, G. ; Pouwelse, J. : Gromit: Benchmarking the Performance and Scalability of Blockchain Systems. In: *2022 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*. IEEE. – ISBN 978–1–66549–172–3, 56–63

[57] Oliveira, M. T. ; Carrara, G. R. ; Fernandes, N. C. ; Albuquerque, C. V. N. ; Carrano, R. C. ; Medeiros, D. S. V. ; Mattos, D. M. F.: Towards a Performance Evaluation of Private Blockchain Frameworks using a Realistic Workload. In: *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, IEEE. – ISBN 978–1–5386–8336–1, 180–187

[58] Pohl, K. ; Rupp, C. : *Basiswissen Requirements Engineering: Aus- und Weiterbildung nach IREB-Standard zum Certified Professional for Requirements Engineering Foundation Level*. 5., überarbeitete und aktualisierte Auflage. dpunkt.verlag https://content-select.com/de/portal/media/view/603e7138-368c-40c9-88ac-05bfb0dd2d03. – ISBN 978–3–86490–814–9

[59] Raghunathan, T. E.: Synthetic Data. 8, Nr. 1, 129–140. http://dx.doi.org/10.1146/annurev-statistics-040720-031848. – DOI 10.1146/annurev–statistics–040720–031848. – ISSN 2326–8298, 2326–831X

[60] Schäffer, M. ; Di Angelo, M. ; Salzer, G. : Performance and Scalability of Private Ethereum Blockchains. Version: 2019. http://dx.doi.org/10.1007/978-3-030-30429-4_8. In: Di Ciccio, C. (Hrsg.) ; Gabryelczyk, R. (Hrsg.) ; García-Bañuelos, L. (Hrsg.) ; Hernaus, T. (Hrsg.) ; Hull, R. (Hrsg.) ; Indihar Štemberger, M. (Hrsg.) ; Kő, A. (Hrsg.) ; Staples, M. (Hrsg.): *Business Process Management: Blockchain and Central and Eastern Europe Forum* Bd. 361. Springer International Publishing. – DOI 10.1007/978–3–030–30429–4_8. – ISBN 978–3–030–30428–7 978–3–030–30429–4, 103–118. – Series Title: Lecture Notes in Business Information Processing

[61] Sigwart, M. ; Borkowski, M. ; Peise, M. ; Schulte, S. ; Tai, S. : A secure and extensible blockchain-based data provenance framework for the Internet of Things. In: *Personal and Ubiquitous Computing* (2020), Jun. http://dx.doi.org/10.1007/s00779-020-01417-z. – DOI 10.1007/s00779–020–01417–z. – ISSN 1617–4909, 1617–4917

[62] Song, Z. ; Yang, L. ; Gaoyang, L. ; Han, Y. ; Baozhong, H. ; Jinwei, S. ; Jingang, F. : An Improved Data Provenance Framework Integrating Blockchain and PROV Model. In: *2020 International Conference on Computer Science and Management Technology (ICCSMT)*. IEEE. – ISBN 978–1–72818–668–9, 323–327

[63] Ucbas, Y. ; Eleyan, A. ; Hammoudeh, M. ; Alohaly, M. : Performance and Scalability Analysis of Ethereum and Hyperledger Fabric. In: *IEEE Access* 11 (2023), 67156–67167. http://dx.doi.org/10.1109/ACCESS.2023.3291618. – DOI 10.1109/ACCESS.2023.3291618. – ISSN 2169–3536

[64] Wang, R. ; Ye, K. ; Meng, T. ; Xu, C.-Z. : Performance Evaluation on Blockchain Systems: A Case Study on Ethereum, Fabric, Sawtooth and Fisco-Bcos. Version: 2020. http://dx.doi.org/10.1007/978-3-030-59592-0_8. In: Wang, Q. (Hrsg.) ; Xia, Y. (Hrsg.) ; Seshadri, S. (Hrsg.) ; Zhang, L.-J. (Hrsg.): *Services Computing – SCC 2020* Bd. 12409. Springer International Publishing. – DOI 10.1007/978–3–030–59592–0_8. – ISBN 978–3–030–59591–3 978–3–030–59592–0, 120–134. – Series Title: Lecture Notes in Computer Science

[65] Wang, R. ; Ye, K. ; Wang, Y. ; Xu, C.-Z. : xBCBench: A Benchmarking Tool for Analyzing the Performance of Blockchain Systems. Version: 2021. http://dx.doi.org/10.1007/978-981-16-7993-3_8. In: Dai, H.-N. (Hrsg.) ; Liu, X. (Hrsg.) ; Luo, D. X. (Hrsg.) ; Xiao, J. (Hrsg.) ; Chen, X. (Hrsg.): *Blockchain and Trustworthy Systems* Bd. 1490. Springer Singapore. – DOI 10.1007/978–981–16–7993–3_8. – ISBN 9789811679926 9789811679933, 101–114. – Series Title: Communications in Computer and Information Science

[66] Yorkston, K. : *Performance Testing: An ISTQB Certified Tester Foundation Level Specialist Certification Review*. Apress. http://dx.doi.org/10.1007/978-1-4842-7255-8. http://dx.doi.org/10.1007/978-1-4842-7255-8. – ISBN 978–1–4842–7254–1 978–1–4842–7255–8

[67] Zhang, W. ; Anand, T. : *Blockchain and Ethereum Smart Contract Solution Development: Dapp Programming with Solidity*. Apress. http://dx.doi.org/10.1007/978-1-4842-8164-2. http://dx.doi.org/10.1007/978-1-4842-8164-2. – ISBN 978–1–4842–8163–5 978–1–4842–8164–2

[68] Zhang, Z. ; Cui, C. ; Tao, L. ; Wang, J. ; Li, D. ; Wu, S. ; Feng, Q. ; Yang, Q. ; Chen, J. ; Zhang, J. ; Zhang, P. ; Zhang, Z. : Research on Consistency Tracing Technology of Dispatching Control Model Data based on Blockchain. In: *2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. IEEE. – ISBN 978–1–72818–028–1, 234–239

[69] Zheng, P. ; Zheng, Z. ; Luo, X. ; Chen, X. ; Liu, X. : A detailed and real-time performance monitoring framework for blockchain systems. In: *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, ACM. – ISBN 978–1–4503–5659–6, 134–143

Tabellenverzeichnis

Abbildungsverzeichnis

PROVN-Datei von Provenance-Graph in Abbildung 5.1

```
document
  prefix userdata <http://software.dlr.de/qs/user/data/>
  prefix graphic <http://software.dlr.de/qs/graphic/>
  prefix device <http://software.dlr.de/qs/device/>
  prefix user <http://software.dlr.de/qs/user/>
  prefix service <http://software.dlr.de/qs/service/>
  prefix qs <http://software.dlr.de/qs/>
  prefix software <http://software.dlr.de/qs/software/>

  activity(qs:start0, 2024-06-03T12:04:15.880607, 2024-06-04T12
    ↪ :04:15.880607)
  agent(user:test1, [prov:type='prov:Empty'])
  activity(qs:test2, 2024-06-03T12:04:15.880607, 2024-06-04T12
    ↪ :04:15.880607)
  agent(user:test3, [prov:type='prov:Empty'])
  entity(userdata:test4, [prov:type='prov:Empty'])
  entity(userdata:test5, [prov:type='prov:Empty'])
  activity(qs:test6, 2024-06-03T12:04:15.880607, 2024-06-04T12
    ↪ :04:15.880607)
  entity(userdata:test7, [prov:type='prov:Empty'])
  entity(userdata:test8, [prov:type='prov:Empty'])
  wasInfluencedBy(qs:start0, user:test1)
  wasInfluencedBy(user:test1, qs:test2)
  actedOnBehalfOf(user:test1, user:test3, -)
  wasInfluencedBy(user:test1, userdata:test4)
  wasStartedBy(qs:test2, userdata:test5, -, -)
  wasInfluencedBy(user:test3, qs:test6)
  hadMember(userdata:test5, userdata:test7)
  wasDerivedFrom(userdata:test5, userdata:test8, -, -, -)
endDocument
```

Konsens	Szenario	TX	Graphgröße [KB]	1n1s	2n1s	4n1s	8n1s	16n1s
PoA	w1-3	20	0,5	7,12	7,12	7,06	6,9	7,28
PoA	w1-3	20	1	7,84	7,89	7,81	7,81	7,7
PoA	w1-3	20	1,5	7,91	7,83	7,81	7,8	7,63
PoA	w1-3	20	2	7,86	7,87	7,82	7,76	7,73
PoA	w1-3	20	2,5	7,93	7,92	7,85	7,81	7,73
PoA	w1-1	20	10	7,28	7,27	7,31	7,03	7,97
PoA	w1-1	20	100	8,77	8,74	8,84	8,87	8,86
PoA	w1-2	20	500	10,93	10,93	11,38	12,28	12,48
PoA	w1-1	20	1000	14,94	14,85	15,82	18,13	19,36

Tabelle 1: Latenz - Ergebnisse für Besu (PoA) - Ein Signer

Konsens	Szenario	TX	Graphgröße	1n1s	2n2s	4n4s	8n8s	16n16s
PoA	w1-3	20	0,5	7,12	7,14	7,04	6,9	7,45
PoA	w1-3	20	1	7,84	7,84	7,75	7,82	7,87
PoA	w1-3	20	1,5	7,91	7,89	7,84	7,78	7,43
PoA	w1-3	20	2	7,86	7,83	7,87	7,7	7,5
PoA	w1-3	20	2,5	7,93	7,87	7,89	7,77	7,59
PoA	w1-1	20	10	7,28	7,29	8,91	7,98	7,82
PoA	w1-1	20	100	8,77	8,83	9,17	9,55	10,2
PoA	w1-2	20	500	10,93	11,82	13,49	15,99	
PoA	w1-1	20	1000	14,94	19,37	21,31	27,07	

Tabelle 2: Latenz - Ergebnisse für Besu (PoA) - Alle Knoten signieren

		1n1s		2n1s		4n1s		8n1s	
TX	Größe	rate	tps	rate	tps	rate	tps	rate	tps
2000	0,5	120	46	100	35,2	50	36,4	50	35,6
1000	1	60	32,1	35	24,3	35	23,3	30	18,1
666	1,5	100	26,6	30	16,8	30	17,3	15	12,7
500	2	80	19,3	40	17	25	13,9	15	10
400	2,5	40	14,8	25	11,9	25	12,5	15	8
100	10	30	5,4	20	5	20	4,7	25	4,2
10	100	1	0,9	5	0,8	1,1	0,5	1,6	0,5
10	1000	0,4	0,1	1,1	0,1	0,4	0,1	0,4	0,1

Tabelle 3: Durchsatz Ergebnisse mit Besu (PoA)

Testnetz	Skalierungsfunktion
1n1s	f(x) = 0,007107x + 7,83289
2n1s	f(x) = 0,006967x + 7,88303
4n1s	f(x) = 0,0114x + 7,36859
8n1s	f(x) = 0,01033x + 7,79967
16n1s	f(x) = 0,01167x + 7,68832
2n2s	f(x) = 0,01154x + 7,828458
4n4s	f(x) = 0,01357x + 7,7364
8n8s	f(x) = 0,019269x + 7,80073
16n16s	f(x) = 0,023535x + 7,84646

Tabelle 4: Skalierungsfunktionen zur Bestimmung der Latenz

Graphgröße	**1n1s**	**2n2s**	**4n4s**	**8n8n**	**16n1s**
0,5	1,74	1,27	1,51	1,47	1,74
1	1,87	1,31	1,46	1,3	1,3
1,5	1,17	1,22	1,32	1,67	1,63
2	1,02	1,13	1,25	1,11	1,03
2,5	0,96	0,97	1,2	1,26	1,25
10	1,69	2,37	1,78	2,62	1,43
100	5,85	4,92	10,66	11,13	5
500	9,19	7,28	19,06	39,83	8,04
1000	20,05	16,14	44,34	95,98	17,43

Tabelle 5: Besu (PoA): Durchschnittliche CPU Auslastung

Graphgröße	**1n1s**	**2n2s**	**4n4s**	**8n8n**	**16n1s**
0,5	89,2	66	77,9	87,7	79,2
1	75,1	66,2	84,8	90,9	79,7
1,5	79,1	65,9	82,4	88,9	79,2
2	80,2	66,4	81,9	85,3	80,3
2,5	81	68,9	81,9	90,4	84,5
10	91,9	69,3	81,4	76,4	82,9
100	269	197	232	254	264
500	370	312	583	622	385
1000	405	434	499	851	450

Tabelle 6: Besu (PoA): Durchschnittlicher RAM-Bedarf

Besu - genesis.json

```
{
    "config": {
      "chainId": 123456,
      "berlinBlock": 0,
      "contractSizeLimit": 2147483647,
      "clique": {
        "blockperiodseconds": 15,
        "epochlength": 30000
      }
    },
    "coinbase": "0x0000000000000000000000000000000000000000",
    "difficulty": "0x1",

    "extraData": "0x0000000000000000000000000000000000000000000000000
       ↪ 00000000000000c7ab6e726d23293eb6581dbc435d464e290019a4
       ↪ 0000000000000000000000000000000000000000000000000000000000
       ↪ 0000000000000000000000000000000000000000000000000000000000
       ↪ 0000000000000000",
    "gasLimit": "0x1fffffffffffff",
    "mixHash": "0x0000000000000000000000000000000000000000000000000
       ↪ 000000000000000",
    "nonce": "0x0",
    "timestamp": "0x5c51a607",
    "alloc": {
      "c7ab6e726d23293eb6581dbc435d464e290019a4": {
        "privateKey": "e5a5d5971acfc08668707
           ↪ b4c292132cc2c3d4d4c7625977ad9f09706f25f8df2",
        "balance": "900000000000000000000000"
      },
      "0e699b128677d9163131a4ee5c7d638a9770b7d0": {
        "privateKey": "d708fce9af769f6c3dd8f59ad9ac1122abecfdeebab6
           ↪ 056b79f185cc57373ba5",
        "balance": "900000000000000000000000"
      },
      "9294a1ac1d647a2522ba1a085c6b89298fbd3b60": {
        "privateKey": "67e2bc75dd4559d37c8ecf4a469b96118e19d4703
           ↪ f798b2e220045abdbd50c79",
        "balance": "900000000000000000000000"
      },
      "7619a24d425f754bf3e661f4a59762979bd32cc9": {
          "privateKey": "0ca4c58e960dffd88e05025b6850137afee994e704
             ↪ cc9dbc7cebf7226f45f99b",
          "balance": "900000000000000000000000"
      },
      "14133a7416220f6659b7cd7771fef58fd63ad2fc": {
```

```
            "privateKey": "e4f8ade209d7895cabd15863a658b442bb3d0
               ↪ fe92924ef6ea50d084f84cb14e0",
            "balance": "900000000000000000000000"
        },
        "e818aa7d02a7cd5d061b38db72e652538bf9f75f": {
            "privateKey": "82d637d9b4644b003
               ↪ cb5ec2bb8749723843f42944484c75051104e3de0858fad",
            "balance": "900000000000000000000000"
        },
        "a5da89ebbf20a4efca427a22deac187ae963fb92": {
            "privateKey": "60d061692374
               ↪ d2c698334e21a4795b3a26731f16d136e4ac7ff19388bf9ae198
               ↪ ",
            "balance": "900000000000000000000000"
        },
        "2adaece020c535b685678f8fd0c134ed5771f41d": {
            "privateKey": "01531b77bcb61a5c7b3a40b336fe084777
               ↪ e8dc58e4c1a8789def7df7b1fc35b6",
            "balance": "900000000000000000000000"
        }
    }
}
```

Besu - runNode1.sh

```
export BESU_OPTS=-Xmx6g

besu --data-path=node1/data \
    --genesis-file=genesis.json \
    --network-id 123456 \
    --rpc-http-enabled=true \
    --rpc-http-port=8545 \
    --rpc-ws-enabled=true \
    --rpc-ws-port=8745 \
    --rpc-http-api=ETH,NET,CLIQUE,ADMIN,TXPOOL  \
    --rpc-ws-api=ETH,NET,CLIQUE,ADMIN,TXPOOL \
    --host-allowlist="*" \
    --rpc-http-cors-origins="all" \
    --rpc-http-host=0.0.0.0 \
    --sync-mode=FULL \
    --min-gas-price=0 \
    --rpc-tx-feecap=0 \
    --discovery-enabled=false \
    --metrics-enabled=true \
    --metrics-port=9545 \
    --poa-block-txs-selection-max-time=+1000 \
    --tx-pool-max-future-by-sender=5000
```

Besu - clean.sh

```
rm -r -f ./node1/data/caches
rm -r -f ./node1/data/database
rm -f ./node1/data/VERSION_METADATA.json
rm -f ./node1/data/DATABASE_METADATA.json

rm -r -f ./node2/data/caches
rm -r -f ./node2/data/database
rm -f ./node2/data/VERSION_METADATA.json
rm -f ./node2/data/DATABASE_METADATA.json

rm -r -f ./node3/data/caches
rm -r -f ./node3/data/database
rm -f ./node3/data/VERSION_METADATA.json
rm -f ./node3/data/DATABASE_METADATA.json

rm -r -f ./node4/data/caches
rm -r -f ./node4/data/database
rm -f ./node4/data/VERSION_METADATA.json
rm -f ./node4/data/DATABASE_METADATA.json

rm -r -f ./node5/data/caches
rm -r -f ./node5/data/database
rm -f ./node5/data/VERSION_METADATA.json
rm -f ./node5/data/DATABASE_METADATA.json

rm -r -f ./node6/data/caches
rm -r -f ./node6/data/database
rm -f ./node6/data/VERSION_METADATA.json
rm -f ./node6/data/DATABASE_METADATA.json

rm -r -f ./node7/data/caches
rm -r -f ./node7/data/database
rm -f ./node7/data/VERSION_METADATA.json
rm -f ./node7/data/DATABASE_METADATA.json

rm -r -f ./node8/data/caches
rm -r -f ./node8/data/database
rm -f ./node8/data/VERSION_METADATA.json
rm -f ./node8/data/DATABASE_METADATA.json
```

besu_propose_signers.py

```
'''
Script to add all peers to every node
'''

import requests
import json
import time
import sys


def get_signers():
    # curl -X POST --data '{"jsonrpc":"2.0","method":"
        ↪ clique_getSigners","params":["latest"], "id":1}'
        ↪ 0.0.0.0:8545
    payload_get = '{"jsonrpc":"2.0","method":"clique_getSigners","p
        ↪ arams":["latest"],"id":1}'
    r = requests.post(url_without_port + ':' + str(first_http_port)
        ↪ , data=payload_get, headers=headers)

    return json.loads(r.text)['result']


def propose_signers(addr):
    propose_counter = 1
    current = set()

    print('\nto propose: ' + str(addr))

    for port in range(first_http_port, first_http_port + nodes):
        counter = 0

        for address in addr[:nodes]:
            if counter == propose_counter:
                break
            else:
                current.add(str(address))
                payload = json.loads(payload_template)
                payload['params'] = [address, True]
                r = requests.post(url_without_port + ':' + str(port
                    ↪ ), data=json.dumps(payload), headers=headers)
                # just the first address
                print(str(address) + ' -> ' + str(port))
                counter += 1
    print('\ncurrent: ' + str(current))


# example curl -s -H 'content-type:application/json' -X POST --data
    ↪  '{"jsonrpc":"2.0","method":"admin_addPeer","params":["enode
```

```
    ↪ ://fb750971a8bbcb8968e1b36a998476
    ↪ e59b11c6182f20894b89f79de8299a36
    ↪ c299fcc72750eb4887377c615c28e6e34cda23e05b626cfbdbdb82d66194a
    ↪ 873d4@127.0.0.1:30318"],"id":1}' 0.0.0.0:8545

nodes = int(sys.argv[1])
first_http_port = 8545

addresses = [
    "0xc7ab6e726d23293eb6581dbc435d464e290019a4",
    "0x0e699b128677d9163131a4ee5c7d638a9770b7d0",
    "0x9294a1ac1d647a2522ba1a085c6b89298fbd3b60",
    "0x7619a24d425f754bf3e661f4a59762979bd32cc9",
    "0x14133a7416220f6659b7cd7771fef58fd63ad2fc",
    "0xe818aa7d02a7cd5d061b38db72e652538bf9f75f",
    "0xa5da89ebbf20a4efca427a22deac187ae963fb92",
    "0x2adaece020c535b685678f8fd0c134ed5771f41d",
    "0x6004037afdf5a89a72eb75b69684485d6af8ab94",
    "0x3dfb4c6868d12f49183f1fa5c449b3bb821bf7db",
    "0x1f8850dbfd5333dbce3e6fbac0cc32cae76091ef",
    "0xa01f9d81c438b4cd6e445ce299b052d7c12b2767",
    "0x19cbf944df690715e4b376803f506df8f5dd06a3",
    "0xd73c3877c8d3ceb0501a4fea52917ba922e7da28",
    "0x8b091a0b36ba425f79edcfbdbb20cf4aa0fd5444",
    "0x81e7536c9862e9f083785816c708b89e0df21b41"
]

url_without_port = 'http://0.0.0.0'
payload_template = '{"jsonrpc":"2.0","method":"clique_propose","par
    ↪ ams":[],"id":1}'
headers = {'content-type': 'application/json', 'Accept-Charset': '
    ↪ UTF-8'}

signers = get_signers()

while len(signers) != nodes:
    print('--------')
    print('total: ' + str(len(signers)) + ', signers: ' + str(
        ↪ signers))
    no_signers = set(addresses[:nodes]) - set(signers)
    propose_signers(list(no_signers))
    time.sleep(15)
    signers = get_signers()

print('Complete! Total: ' + str(len(signers)) + ', signers: ' + str
    ↪ (signers))
```

besu_networkconfig.json

```
{
    "caliper": {
        "blockchain": "ethereum"
    },
    "ethereum": {
        "url": "ws://localhost:8745",
        "contractDeployerAddress": "0xc7ab6e726d23293eb6581
            ↪ dbc435d464e290019a4",
        "contractDeployerAddressPrivateKey": "0xe5a5d5971
            ↪ acfc08668707b4c292132
            ↪ cc2c3d4d4c7625977ad9f09706f25f8df2",
        "fromAddress": "0xc7ab6e726d23293eb6581dbc435d464e290019a4"
            ↪ ,
        "fromAddressPrivateKey": "0xe5a5d5971acfc08668707b4c292132
            ↪ cc2c3d4d4c7625977ad9f09706f25f8df2",
        "transactionConfirmationBlocks": 2,
        "contracts": {
            "provStorageAutoIndex": {
                "path": "./src/ethereum/ma/provStorageAutoIndex.
                    ↪ json",
                "estimateGas": true,
                "gas": {
                    "createProvenance": 1200000000000,
                    "getProvenance": 100000000
                }
            }
        }
    }
}
```

geth_networkconfig.json

1

```
{
    "caliper": {
        "blockchain": "ethereum"
    },
    "ethereum": {
        "url": "ws://localhost:8845",
        "contractDeployerAddress": "0
            ↪ x123463a4b065722e99115d6c222f267d9cabb524",
        "contractDeployerAddressPassword": "test",
        "fromAddress": "0x123463a4b065722e99115d6c222f267d9cabb524"
            ↪ ,
```

```
        "fromAddressPassword": "test",
        "transactionConfirmationBlocks": 2,
        "contracts": {
            "provStorageAutoIndex": {
                "path": "./src/ethereum/ma/provStorageAutoIndex.
                    ↪ json",
                "estimateGas": true,
                "gas": {
                    "createProvenance": 1200000000000,
                    "getProvenance": 100000000
                }
            }
        }
    }
}
```

provStorageAutoIndex.json

1

```
{
    "name": "provStorageAutoIndex",
    "abi": [{"inputs":[{"internalType":"string","name":"_hash","
        ↪ type":"string"},{"internalType":"string","name":"_context
        ↪ ","type":"string"}], ...
    }],
    "bytecode": "6080604052...",
    "gas": 200000000
}
```

Geth - genesis.json

```
{
    "config": {
      "chainId": 12345,
      "homesteadBlock": 0,
      "eip150Block": 0,
      "eip155Block": 0,
      "eip158Block": 0,
      "byzantiumBlock": 0,
      "constantinopleBlock": 0,
      "petersburgBlock": 0,
      "istanbulBlock": 0,
      "berlinBlock": 0,
      "clique": {
        "period": 15,
```

```
        "epoch": 30000
      }
    },
    "difficulty": "0x1",
    "gasLimit": "0x1C9C380",
    "extradata": "0x0000000000000000000000000000000000000000000000000
      ↪ 0000000000000000CC28f815292dAb8D5755a696fb0f2020b570216c
      ↪ 0000000000000000000000000000000000000000000000000000000000000000
      ↪ 0000000000000000000000000000000000000000000000000000000000000000
      ↪ 0000000000000000",
    "alloc": {
      "CC28f815292dAb8D5755a696fb0f2020b570216c": { "balance": "900
        ↪ 000000000000000000000" },
      "0422dAB98FbD0DeBd8786574Ee9b1757faBB9928": { "balance": "900
        ↪ 000000000000000000000" },
      "23ac60c39393F5CB696c2ee2C601afcE2faEde83": { "balance": "900
        ↪ 000000000000000000000" },
      "856134b919d0d7B5d6ed157e48E946768771E75F": { "balance": "900
        ↪ 000000000000000000000" }
    }
  }
```

Geth - runNode1.sh

```
./geth --datadir node1 \
    --port 30409 \
    --bootnodes enode://4b645fb351963251648d6d7942b6674ee2a01eaf894
      ↪ 050b57d08b0840ec8c921be8703ce894e0e1cde1c40e63d3c2c557b88
      ↪ 01836dc285dc94f0d1e4f7a4e488@127.0.0.1:0?discport=30408 \
    --networkid 12345 \
    --unlock 0xCC28f815292dAb8D5755a696fb0f2020b570216c \
    --password pw.txt \
    --authrpc.port 8571 \
    --ws.api admin,eth,miner,personal,web3 \
    --ws \
    --ws.port 8745 \
    --mine \
    --miner.etherbase 0xCC28f815292dAb8D5755a696fb0f2020b570216c \
    --http \
    --http.port 3338 \
    --allow-insecure-unlock \
    --rpc.gascap 0 \
    --rpc.txfeecap 0 \
    --miner.gasprice 0 \
    --http.corsdomain '*' \
    --http.addr 0.0.0.0
```

provStorageAutoIndex.sol

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.8.2;

contract ProvStorageAutoIndex {
    struct ProvenanceRecord {
        string hash;
        string context;
        uint index;
    }

    mapping (uint => ProvenanceRecord) private provenanceRecords;
    uint[] private provenanceIndex;

    function isStored(uint _provId) public view returns (bool
       ↪ exists) {
        if (provenanceRecords[_provId].index >= provenanceIndex.
           ↪ length) {
            return false;
        } else {
            return (provenanceIndex[provenanceRecords[_provId].
               ↪ index] == _provId);
        }
    }

    function getProvenance(uint _provId) public view returns (
       ↪ string memory hash, string memory context, uint index) {
        require(isStored(_provId), "Record does not exist");

        return (
            provenanceRecords[_provId].hash,
            provenanceRecords[_provId].context,
            provenanceRecords[_provId].index
        );
    }

    function createProvenance(string memory _hash, string memory
       ↪ _context) public returns (uint index) {
        //require(!isStored(_provId), "Record already exists");
        uint _provId = 0;

        if (provenanceIndex.length != 0) {
            _provId = provenanceIndex[provenanceIndex.length-1] +
               ↪ 1;
        }
```

```
        provenanceIndex.push(_provId);

        provenanceRecords[_provId].hash = _hash;
        provenanceRecords[_provId].context = _context;
        provenanceRecords[_provId].index = provenanceIndex.length -
            ↪ 1;

        return provenanceRecords[_provId].index;
    }

    function updateProvenance(uint _provId, string memory _hash,
        ↪ string memory _context) public returns (bool success) {
        require(isStored(_provId), "Record does not exist");

        provenanceRecords[_provId].hash = _hash;
        provenanceRecords[_provId].context = _context;

        return true;
    }

    function getProvenanceCount() public view returns (uint count)
        ↪ {
        return provenanceIndex.length;
    }

    function getProvenanceIdAtIndex( uint _index) public view
        ↪ returns (uint provId) {
        require(_index < provenanceIndex.length, "Index out of
            ↪ bounds");
        return provenanceIndex[_index];
    }

    function deleteProvenance(uint _provId) public returns (bool
        ↪ success) {
        require(isStored(_provId), "Record does not exist");

        uint recordToDelete = provenanceRecords[_provId].index;
        uint keyToMove = provenanceIndex[provenanceIndex.length-1];

        // Overwrite index of record to be deleted with key of last
            ↪ provenance record in index, update index in last
            ↪ provenance record
        provenanceIndex[recordToDelete] = keyToMove;
        provenanceRecords[keyToMove].index = recordToDelete;
        provenanceIndex.pop();

        return true;
    }
```

```
}
```

deploy_contract.py

```
# contract has to be compiled with solc
from web3 import Web3
from web3.middleware import geth_poa_middleware

bin_name = 'provStorageAutoIndex.bin'
abi_name = 'provStorageAutoIndex.abi'
contract_file = 'provStorageAutoIndex.sol'
folder = r'../contract/'

abi = ''
bytecode = ''

with open(folder + bin_name, 'r') as file:
    bytecode = '0x' + file.read()

with open(folder + abi_name, 'r') as file:
    abi = file.read()

# rpc node
w3 = Web3(Web3.HTTPProvider('http://127.0.0.1:3338'))
w3.middleware_onion.inject(geth_poa_middleware, layer=0)

w3.eth.default_account = w3.eth.accounts[0]

Contract = w3.eth.contract(abi=abi, bytecode=bytecode)
tx_hash = Contract.constructor().transact()

print('tx hash: ' + tx_hash.hex())

tx_receipt = w3.eth.wait_for_transaction_receipt(tx_hash)

print('contract address: ' + tx_receipt.contractAddress)
```

test_contract.py

```
from web3 import Web3
from web3.middleware import geth_poa_middleware
from hashlib import sha256


def print_tx_receipt(tx_receipt):
```

```
    print('tx info:')
    print('---------')
    print('status: ' + str(tx_receipt.status))
    print('block number: ' + str(tx_receipt.blockNumber))
    print('transactionHash: ' + tx_receipt.transactionHash.hex())
    print('cumulativeGasUsed: ' + str(tx_receipt.cumulativeGasUsed)
        ↪ )
    print('gasUsed: ' + str(tx_receipt.gasUsed))


def store_prov_rec(id, hash, content):
    tx_hash = prov_store.functions.createProvenance(id, hash,
        ↪ content).transact()
    print('tx hash: ' + tx_hash.hex())
    tx_receipt = w3.eth.wait_for_transaction_receipt(tx_hash)

    return tx_receipt


def store_prov_rec_auto(hash, content):
    tx_hash = prov_store.functions.createProvenance(hash, content).
        ↪ transact()
    print('tx hash: ' + tx_hash.hex())
    tx_receipt = w3.eth.wait_for_transaction_receipt(tx_hash)

    return tx_receipt


def get_prov_rec(id):
    return prov_store.functions.getProvenance(id).call()


def load_testdata():
    folder = r'../testdata/'

    input_file = {
        'kb_0.5': 'kb_0.5_graph.provn',
        'kb_1': 'kb_1_graph.provn',
        'kb_1.5': 'kb_1.5_graph.provn',
        'kb_2': 'kb_2_graph.provn',
        'kb_2.5': 'kb_2.5_graph.provn',
        'kb_10': 'kb_10_graph.provn',
        'kb_13': 'kb_13_graph.provn',
        'kb_14': 'kb_14_graph.provn',
        'kb_15': 'kb_15_graph.provn',
        'kb_41': 'kb_41_graph.provn',
        'kb_42': 'kb_42_graph.provn',
        'kb_43': 'kb_43_graph.provn',
```

```
        'kb_44': 'kb_44_graph.provn',
    }

    testdata = {
        'kb_0.5': '',
        'kb_1': '',
        'kb_1.5': '',
        'kb_2': '',
        'kb_2.5': '',
        'kb_10': '',
        'kb_10': '',
        'kb_41': '',
        'kb_42': '',
        'kb_43': '',
        'kb_44': '',
    }

    for key in input_file:
        with open(folder + input_file[key], 'r') as file:
            testdata[key] = file.read()

    return testdata

folder = r'../contract/'
#abi_name = 'provStorage.abi'
abi_name = 'provStorageAutoIndex.abi'
abi = ''
contract_adresss = '0xAbB1D65502614d20959bA2c8520f96442F9E86a5'

with open(folder + abi_name, 'r') as file:
    abi = file.read()

#w3 = Web3(Web3.IPCProvider("/home/twigm/poa_testnets/
    ↪ besu_poa_devnet/node1/data/besu.ipc"))
# rpc node
w3 = Web3(Web3.HTTPProvider('http://127.0.0.1:3338'))
w3.middleware_onion.inject(geth_poa_middleware, layer=0)
#w3.eth.default_account = w3.eth.accounts[0]
w3.eth.default_account = '0xCC28f815292dAb8D5755a696fb0f2020b570216
    ↪ c'

prov_store = w3.eth.contract(
    address=contract_adresss,
    abi=abi
)

testdata = load_testdata()
```

```
h = sha256(testdata['kb_14'].encode('utf-8')).hexdigest()
tx_receipt = store_prov_rec_auto(h, testdata['kb_14'])
#tx_receipt = store_prov_rec_auto('a', 'test')
print_tx_receipt(tx_receipt)

#tx_receipt = store_prov_rec(101, 'zz', 'test')
#print_tx_receipt(tx_receipt)


#print(get_prov_rec(101))
```

...