FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

A Framework for Blockchain Interoperability and Runtime Selection

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Ing. Philipp Frauenthaler, BSc
Matrikelnummer 01225901

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Privatdoz. Dr.-Ing. Stefan Schulte

Wien, 5. November 2018

Philipp Frauenthaler

Stefan Schulte

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

A Framework for Blockchain Interoperability and Runtime Selection

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Ing. Philipp Frauenthaler, BSc
Registration Number 01225901

to the Faculty of Informatics

at the TU Wien

Advisor: Privatdoz. Dr.-Ing. Stefan Schulte

Vienna, 5th November, 2018

Philipp Frauenthaler	Stefan Schulte

Erklärung zur Verfassung der Arbeit

Ing. Philipp Frauenthaler, BSc
Allachweg 311, 8250 Vorau

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

Wien, 5. November 2018

Philipp Frauenthaler

Acknowledgements

At this place I want to use the opportunity to thank my advisor Stefan Schulte for the excellent support, constructive and fast feedbacks, and precise comments throughout the last eight months.

Furthermore, I want to thank my friend Marten Sigwart for the valuable discussions about blockchain technologies.

Finally, I would like to express my deepest gratitude to my family and especially my girlfriend Corinna Pölzler for their moral encouragements and support.

Kurzfassung

In den letzten Jahren erlangten kryptographische Währungen, auch Kryptowährungen genannt, enorm an Popularität. Die erste und zugleich prominenteste Kryptowährung ist Bitcoin. Bitcoin wurde im Jahr 2008 von Satoshi Nakamoto vorgestellt. In den darauffolgenden Jahren wurden zahlreiche weitere Kryptowährungen gegründet. Im ersten Quartal 2018 existierten bereits über 1.000 kryptographische Währungen. Viele dieser unterschiedlichen Währungen funktionieren ohne eine zentrale Instanz. Stattdessen wird häufig eine Blockchain eingesetzt. Eine Blockchain ist ein verteiltes Bestandsbuch, das von anonymen, über ein Peer-to-Peer-Netzwerk miteinander verbundenen, Benutzern verwaltet wird.

Neben dem Aufzeichnen von Transaktionen in zahlreichen Kryptowährungen ist die Blockchain auch in einer Vielzahl weiterer Anwendungsfälle einsetzbar. Populäre Anwendungsfälle der letzten Jahre sind z. B. die digitale Stimmabgabe, Notariatsdienste, Supply Chain Management, Auditierung und Kontrolle von Eigentumsrechten.

Die Eignung einer bestimmten Blockchain für einen gegebenen Anwendungsfall hängt von mehreren Kriterien ab, z. B. den Kosten für das Schreiben von Daten in diese Blockchain, der Zeit, die es benötigt, bis ein Datensatz permanent in der Blockchain festgeschrieben ist, der Verteilung der Hashpower unter Minern oder Mining Pools oder der Netzwerkhashrate. Diese Kriterien können sich im Laufe der Zeit ändern. Daher kann es passieren, dass eine bestimmte Blockchain in Zukunft für einen gegebenen Anwendungsfall nicht mehr geeignet ist. Derartige Unsicherheiten verringern den praktischen Wert von Blockchains.

Um die Auswirkungen dieser Einschränkung in der praktischen Anwendung zu begrenzen, entwickeln wir in dieser Diplomarbeit ein Framework, das es ermöglicht, zwischen Blockchains zu wechseln. Das Framework überwacht mehrere Blockchains, berechnet den jeweiligen Nutzen und ermittelt auf Basis dieser Berechnungen die nutzbringendste Blockchain. Weiters kann das Framework auf verschiedene Ereignisse wie einer rapiden Abnahme der Netzwerkhashrate oder einen signifikanten Kostenanstieg reagieren. Die Bewertung mehrerer Blockchains, die Reaktion auf verschiedene Ereignisse und das Wechseln zwischen Blockchains ermöglichen es Benutzern, zu einer für sie günstigeren Blockchain zu wechseln, um von niedrigeren Kosten, besserer Performanz oder erhöhter Sicherheit zu profitieren. Die Referenzimplementierung des Frameworks unterstützt Bitcoin, Ethereum, Ethereum Classic und Expanse. Aufgrund des modularen Aufbaus kann das Framework im Bedarfsfall erweitert werden, um bspw. weitere Blockchains zu integrieren.

Abstract

In the past few years, cryptographic currencies, also referred to as cryptocurrencies, gained much popularity. The first and most prominent cryptocurrency is Bitcoin, announced in 2008 by Satoshi Nakamoto. In the first quarter 2018, there have been more than 1,000 cryptocurrencies in existence. The common property of cryptocurrencies is that they are not owned or controlled by a single authority, e.g., by a central bank. In order to eliminate the need for a central entity, many cryptocurrencies use a blockchain for keeping track of payment transactions and state changes. The blockchain is a distributed ledger that is maintained by anonymous users connected via peer-to-peer networks.

Besides keeping track of payment transactions and state changes, blockchains are applicable for a wide range of use cases. Popular use cases that came up in the last years are, e.g., digital voting, notary services, Supply Chain Management, auditing and control of ownership rights.

The suitability of a particular blockchain for a given use case depends on various criteria, e.g., the costs for writing data into that blockchain, the time until a data record is permanently included and thus remains unchanged with high probability, the distribution of the network's hash power among miners or mining pools, the network's hash rate, etc. These properties can vary over the time. Thus, a particular blockchain can become unsuitable for a given use case over time. Such uncertainties can limit the practical value of blockchains.

In order to overcome this limitation, we design and develop a framework that is capable of switching back and forth between blockchains. The framework monitors several blockchains, calculates their individual benefits and determines the most beneficial one. Furthermore, the framework is able to react to various events such as a rapid decrease of a network's hash power or a steadily increase of the costs for writing data into a blockchain. The assessment of several blockchains, the mechanism for reacting to various events and the switchover functionality enable users to switch to another, more beneficial blockchain in order to benefit from low costs, high performance or better security. The reference implementation of the proposed framework supports Bitcoin, Ethereum, Ethereum Classic and Expanse. The modular design of the framework allows future researchers to extend the framework, e.g., to add support for further blockchains.

Contents

CHAPTER 1

Introduction

1.1 Motivation and Aim of the Work

In the past few years, cryptographic currencies, also referred to as cryptocurrencies, gained much popularity [Dzi15]. The first and most prominent cryptocurrency is Bitcoin, announced in 2008 by Satoshi Nakamoto [BMC$^+$15a, Nak08, TS16]. At the end of 2017, Bitcoin had a market capitalization of more than 200 billion USD [Mar18]. Figure 1.1 shows the growth of the market capitalization of Bitcoin between 2012 and the first quarter 2018. After the launch of Bitcoin in 2009, further cryptocurrencies, such as Litecoin, Namecoin and SwiftCoin have been emerged. In the first quarter 2018, there have been more than 1,000 cryptocurrencies in existence. About 40 of them had a market capitalization of more than one billion USD, respectively [Cor18, Fra18].

The common property of cryptocurrencies is that they are not owned or controlled by a single authority, e.g., a central bank [Dzi15]. In order to eliminate the need of a central entity, they are maintained by anonymous users connected via peer-to-peer networks. Bitcoin's design eliminates centralized banks in a sense, that everyone is the bank, i.e., every participant keeps a copy of records that would normally be stored at a bank. This decentralized approach of recording financial information is known as distributed ledger. The distributed ledger keeps track of all payment transactions and ownerships [TS16]. Its distributed architecture, where multiple copies of the entire data are stored on different participating nodes, opens up new possibilities to cheat. In order to prevent users from double spending, i.e., sending the same coin to multiple recipients, the entire network verifies the legitimacy of the transactions. In Bitcoin (and other cryptocurrencies as well), the so-called blockchain takes the role of the distributed ledger [NBF$^+$16, TS16].

The blockchain is a linked list composed of several blocks that include a pointer to the previously validated block in the chain. The pointer is implemented by holding the hash of the preceding block, thus such a pointer is also called hash pointer. Besides the hash

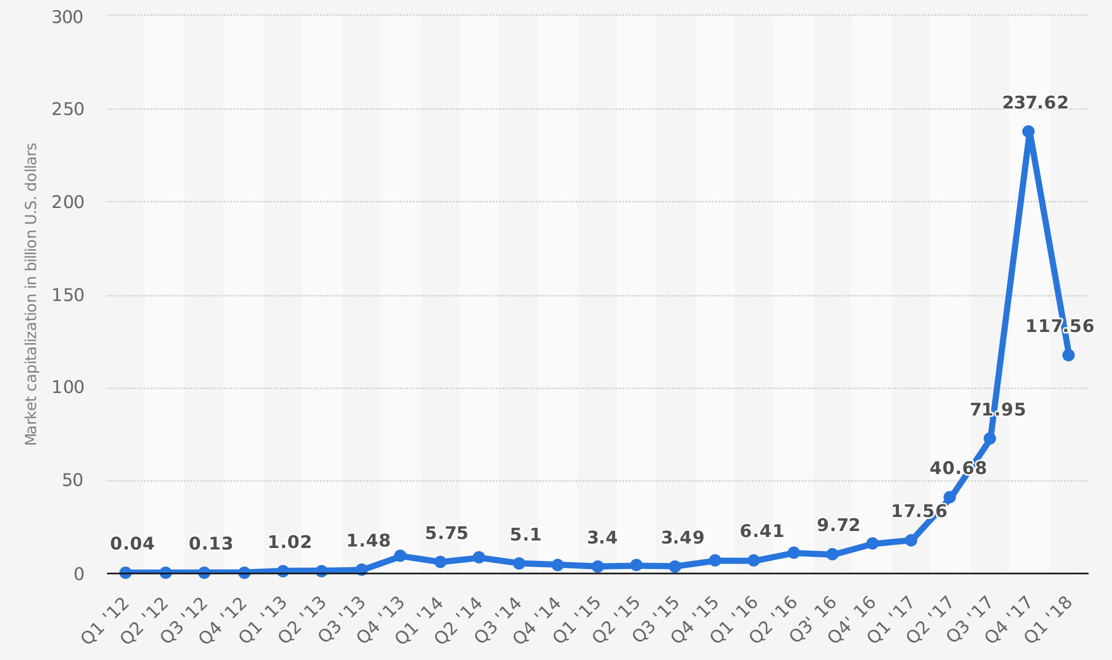


Figure 1.1: Market capitalization of Bitcoin from first quarter 2012 to first quarter 2018 (in billion U.S. dollars) [sta18].

pointer and meta data, a block stores also a certain amount of payment transactions. Overall, the blockchain is a linked list in which each block holds a hash pointer to the previously inserted block and some of the transactions that have been published to the network by cryptocurrency users [NBF+16, TS16]. New blocks are inserted by so-called miners. Miners maintain the blockchain and get paid for their service with a mining reward and transaction fees.

An important aspect of a blockchain is the tamper-evident property. If somebody alters data that is stored in the chain, this alteration can be detected. Due to the utilization of hash pointers, any alteration of a block implies the hash pointers in succeeding blocks to be changed to a new value in order to be compliant with the altered chain. As long as the root hash of the blockchain is stored in a secure place, any alteration can be detected. Assuming an adversary changes the data of some block *k*, the hash pointer in block *k+1*, which is a hash of the entire block *k*, is not going to match up. In order to match up with the hash of the previous block, the adversary must change the hash pointer included in block *k+1* to the new hash value of block *k*. Then, the hash pointer of block *k+2* is not going to match up. Thus, the adversary must change every block that succeeds the altered block in the chain. Since the root hash of the entire blockchain is stored in a secure place, the adversary cannot change it. Therefore, the adversary will be unable to alter any block without being detected [NBF+16].

According to their features and applications, blockchains are categorized into three generations. Blockchain 1.0 is the oldest class of blockchain applications mainly used to store payment transactions. Bitcoin's blockchain belongs to this class. Blockchains of the second generation are characterized by their support for decentralized applications (DApps). They provide a Turing-complete language for writing so-called smart contracts. Smart contracts are programs which are automatically executed by the miners whenever their encoded conditions are triggered. A popular representative of this class is Ethereum. Blockchain 3.0 applications involve the industry and public sector. Furthermore, blockchains belonging to this class are likely to interact with the physical world and therefore applicable for the Internet of Things [ZJ18].

Besides keeping track of payment transactions, blockchains are applicable for a wide range of use cases. Popular use cases that came up in the last years are digital voting, notary services, Supply Chain Management (SCM), auditing, control of ownership rights, cloud storage and many more [Dou18].

A major drawback of the utilization of a blockchain are exchange rate fluctuations[1] that can increase the price for publishing data to a particular blockchain. At some time, the utilization of a certain blockchain (e.g., Bitcoin or Ethereum) might be very cheap and thus a suitable approach. Over the time, this can change very quickly due to the sensitivity of cryptocurrencies for price fluctuations [Kle18, Mon17]. Another important aspect is the duration needed to store data permanently in the blockchain, i.e., how long it takes for a certain piece of data to get mined. This aspect depends highly on the workload and the underlying blockchain technology [The18b]. Since it is difficult to predict the workload a blockchain is confronted with, the growth of the time needed for data to get included in the blockchain is unclear. Furthermore, people might lose trust in a particular blockchain at some time. For instance, this might be the case if many miners switch to another blockchain that is more lucrative for them. The stability of a blockchain and especially a cryptocurrency is not guaranteed in case a single non-compliant miner controls the majority of computational power, since this miner could simply ignore blocks found by other miners and build her own blockchain. If the number of miners decreases, it will become easier to control the majority of the computational power, since the overall computational power decreases as well. Even in the absence of a single miner that controls the majority, smaller miners could collude and form a cartel controlling a majority of the mining power and make use of any strategy that is available to a single majority miner. Mining pools could be a technical mechanism for forming such a cartel [BMC$^+$15b]. A mining pool is a group of miners that attempt to mine a block. Regardless which miner actually finds a block, the pool manager will receive the rewards and will distribute them to all miners of the pool based on the amount of work each miner has performed [NBF$^+$16]. Blockchain technologies such as Ethereum are constantly extended with new features, e.g., Ethereum developers plan to replace the proof-of-work consensus algorithm with the proof-of-stake mechanism [com18b, Ros17]. Consequences of future changes that are fundamental to a system's architecture might be unclear. Furthermore, it is not known if there will be consensus in the community

[1] `https://blockchain.info/charts/market-price`

about future changes. If no community consensus can be reached, the result might be a fork as in the case of Ethereum and Ethereum Classic [Mos17]. On the other side, developers of a particular blockchain might come up with new sophisticated features that are not supported on the currently used blockchain, e.g., Ethereum enables users to write arbitrarily powerful functions, whereas the abilities in Bitcoin are limited, since Bitcoin's scripting language is not Turing-complete [Min17, NBF+16].

Due to the outlined tentativenesses, a solution is required in order to switch to another blockchain on the basis of certain criteria. Since the suitability of a blockchain for a given use case can change rapidly, the solution should be capable of switching back and forth between various blockchains. The amount of data that should be transferred to the destination blockchain might depend on the cause of the intended blockchain switchover. For instance, if the switchover should be performed due to increased costs, it might be sufficient to transfer no data or only a small amount of data to the destination blockchain. In case people are losing trust in the currently used blockchain, it might be essential to transfer all data or at least data of a specific period of time to the destination blockchain. Furthermore, various blockchains should be monitored, operating numbers gathered and according to calculated metrics a suitable destination blockchain should be chosen at runtime. Examples of metrics are storage costs, the average or median block time (i.e., the time it takes to mine a block), and the transaction throughput. Once a suitable blockchain has been selected, the switchover should be initiated.

The goal of this thesis is to address the following research challenges:

RQ1. Which approaches can be used for blockchain interoperability?

The first research challenge aims to provide an overview of various approaches for establishing interoperability between multiple blockchains. This includes an explanation of basic concepts regarding blockchain interoperability and of protocols that enable users of different cryptocurrencies to exchange their tokens in an atomic fashion. Furthermore, we highlight several promising projects in the area of blockchain interoperability and especially atomic cross-chain swaps.

RQ2. Which blockchain metrics are relevant for the runtime selection algorithm?

The second research challenge aims to provide a definition of expressive criteria for selecting a blockchain as destination for the data movement. Furthermore, a parameterizable and weighted ranking specifying the priority of each criterion, and methods for measuring the required criteria are described in this thesis.

RQ3. How can the runtime selection of an appropriate blockchain and the switchover between blockchains be performed?

The third research challenge aims to provide a mechanism for selecting a suitable blockchain on the basis of gathered blockchain metrics and for the switchover between multiple blockchains. We provide a reference implementation that supports the monitoring of several blockchains in order to gather relevant blockchain operating numbers, the calculation of expressive metrics that serve as a basis for the blockchain selection, the selection of an appropriate blockchain and the movement of data to the selected chain.

RQ4. What are the benefits of using the proposed solution?

The fourth research challenge aims to provide an evaluation of the proposed framework in the context of different evaluation scenarios. We report on the benefits of the proposed solution in terms of costs, performance and security.

1.2 Methodology and Approach

The methodological approach followed in this thesis can be structured roughly into five parts that are mostly followed in a linear manner. The following paragraphs summarize these parts:

Literature study In a first step, literature in the area of blockchain interoperability, atomic cross-chain swaps and blockchain metrics will be studied in order to gain a deeper understanding of these fields.

Overview of different approaches suitable for blockchain interoperability On the basis of the gained knowledge, an overview of related work and state-of-the-art methods in the area of blockchain interoperability and cross-chain swaps will be presented. Furthermore, several promising projects will be described.

Definition of blockchain selection criteria and their measurement A definition of expressive criteria will serve as a basis for selecting a suitable blockchain. Furthermore, methods for measuring the required criteria will be described in this thesis.

Development of a framework for blockchain selection and data movement On the basis of the defined selection criteria and a mechanism for switching back and forth between several blockchains, a reference implementation of the proposed solution will be provided.

Evaluation of the developed framework Within the scope of this thesis, it is analyzed whether the developed framework serves the aimed purpose and fulfills the specified requirements. The evaluation will range from the analysis of features and mechanisms (e.g., the selection of the most beneficial blockchain, data movement to another chain, etc.) to more specific assessments of resource consumption and switchover times. The evaluation will conclude with an analysis of the benefits of the developed framework in terms of costs, performance and security.

1.3 Structure of this Thesis

This thesis is structured as follows. Chapter 2 presents background knowledge that is needed to understand the full extent of the proposed solution. The chapter starts off with basic information about the blockchain technology and continues with more advanced topics, e.g., Simplified Payment Verification (SPV). Chapter 3 covers relevant work, state-of-the-art methods and projects in the area of blockchain interoperability. Chapter 4 describes a use case that will be used as reference for further analysis and evaluation of the proposed framework. Chapter 5 introduces the requirements, the design and the implementation of the proposed framework. In Chapter 6, the framework is analyzed and evaluated in the context of different evaluation scenarios. Finally, Chapter 7 concludes the thesis with an overview of the work done and gives an outlook of future work in this area.

CHAPTER 2

Background

The following chapter covers preliminary concepts and theoretical foundations that serve as a basis for the subsequent chapters in this thesis. Section 2.1 gives a brief introduction into cryptographic basics and data structures that are used in Bitcoin or Ethereum. Section 2.2 presents basic concepts of Bitcoin. In Section 2.3, a brief overview of the basic concepts of Ethereum is given. The chapter concludes with a technical comparison between Bitcoin and Ethereum.

2.1 Cryptographic Basics and Data Structures

This section covers basic knowledge about methods for cryptographically securing blockchains and cryptocurrencies.

2.1.1 Cryptographic Hash Functions

The first cryptographic primitive that needs to be introduced is the cryptographic hash function. In general, a hash function is a mathematical function that is defined by the following three properties [NBF$^+$16]:

- The input can be any string of arbitrary size.
- The function generates a fixed-sized output, e.g., 256-bit values.
- The output is efficiently computable, i.e., for a given n-bit input string, the computation of the hash should have a running time that is *O(n)*.

Thus, a general hash function is a function that maps input strings of any size to fixed-sized outputs that can be computed efficiently. For a hash function to be cryptographically secure, it must fulfill additional properties: collision resistance, hiding and puzzle friendliness. A hash function with these additional properties is also referred to as cryptographic hash function. The puzzle-friendliness property is not a general requirement for cryptographic hash functions, but is very useful for cryptocurrencies and blockchains specifically [NBF$^+$16].

Property 1: Collision Resistance

The first property of a cryptographic hash function is the collision resistance. A collision occurs if two distinct input strings produce the same output. A hash function $H(\cdot)$ is collision resistant if it is infeasible to find two input values x and y ($x \neq y$), such that $H(x) = H(y)$. Obviously, it is not impossible to find two distinct inputs that produce the same output, since the input space of the hash function contains all strings of all lengths and the output space contains only strings of a specific fixed length. Because the size of the input space exceeds the size of the output space, it is guaranteed that there must be two distinct inputs that generate the same output. The hash functions used in modern computer systems have a very large input space which makes it infeasible to find a collision. It is still a question of research whether there exists an efficient algorithm to detect collisions for large input spaces. Hash functions that are used in practice have not been proven to be collision-resistant. People have tried very hard to find collisions efficiently and have not yet succeeded. Therefore, engineers assume that cryptographic hash functions are collision resistant [NBF$^+$16].

Property 2: Hiding

The second property of a cryptographic hash function asserts that for a given output of a hash function $y = H(x)$, it is unfeasible to figure out what the input string x was. An important aspect is that this property can only be true if the input strings are spread out. This means that if values from a spread-out set are sampled, there is no certain value that is likely to occur. In order to achieve the hiding property even if the inputs are not spread out, the input strings are concatenated with other inputs that come from a spread-out set [NBF$^+$16].

This property plays an important role in the atomic cross-chain swap (ACCS) protocol outlined in Section 3.1.1. If the hiding property would not hold for the used hash function, a trading participant could steal the other's funds.

Property 3: Puzzle Friendliness

A hash function satisfies the puzzle-friendliness property if for every possible n-bit output y, it is infeasible to find x such that $H(k||x) = y$ in time significantly less than 2^n. k is a value that is chosen from a spread-out set and the double vertical bar || denotes the concatenation of two values. If someone wants to target a hash function that satisfies the puzzle-friendliness property to produce some particular output string y, and if a part of the input has been chosen in a sufficiently randomized way, it is very difficult to find another input string that maps exactly to the target y [NBF$^+$16].

An application that shows the utilization of this property is the *search puzzle*. This is a mathematical problem that requires to find a particular value in a very large space and there is no other solution than searching this large space. The search puzzle consists of

- a hash function $H(\cdot)$,
- the puzzle-ID (abbreviated as id) that is chosen from a spread-out set, and
- a target set Y.

A solution to this puzzle is a input value x, such that $H(id||x) \in Y$. Solving this puzzle means to find an input value such that the produced output falls within Y. The size of Y determines the difficulty of the puzzle. The smaller the size of Y becomes, the harder it is to find a solution. If Y contains only one element, then the puzzle is maximally hard. If the utilized hash function satisfies the puzzle-friendliness property, there is no better solution than just trying random values of x. This mathematical problem plays a key role in the mining of cryptocurrencies such as Bitcoin [NBF+16].

2.1.2 Hash Pointers

Analogous to regular pointers, hash pointers store the location at which some information resides. Additionally, hash pointers contain a cryptographic hash of the information. Due the cryptographic hash, hash pointers also allow to verify that the information has not been changed, whereas a regular pointer only locates the information (Figure 2.1) [NBF+16].

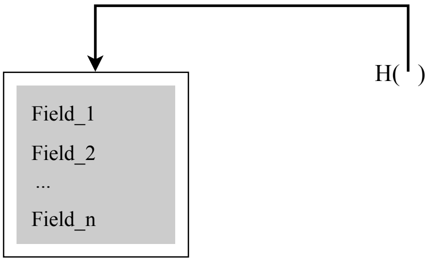


Figure 2.1: A hash pointer is a pointer to the location where a data node is stored together with a cryptographic hash of this node.

2.1.3 Blockchain

In a regular linked list that is composed of a series of blocks, each block contains data and a pointer to the previous block in the list. A blockchain is a linked list that utilizes hash pointers instead of regular pointers (Figure 2.2).

An application of the blockchain is a tamper-evident log, where new data is appended at the end of the log. If somebody alters data that is stored in the chain, this alteration can

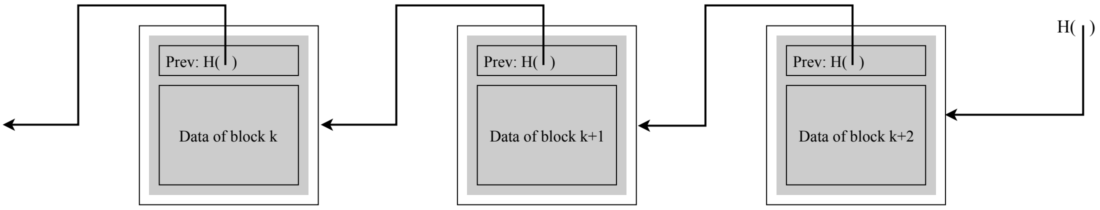


Figure 2.2: A blockchain is a linked list that utilizes hash pointers instead of regular pointers.

be detected. The blockchain achieves the tamper-evident property due to the utilization of hash pointers. Any alteration of a block implies the hash pointers in succeeding blocks to be changed to a new value in order to be compliant with the altered chain. As long as the root hash of the blockchain is stored in a secure place, any alteration can be detected. Assuming an adversary changes the data of some block *k*, the hash pointer in block *k+1*, which is a hash of the entire block *k*, is not going to match up, since the cryptographic hash function is collision-resistant (Figure 2.3). In order to match up with the hash of the previous block, the adversary must change the hash pointer included in block *k+1* to the new hash value of block *k*. Then, the hash pointer of block *k+2* is not going to match up. Thus, the adversary must change every block that succeeds the altered block in the chain. Since the root hash of the entire blockchain is stored in a secure place, the adversary cannot change it. Therefore, the adversary will be unable to alter any block without being detected [NBF+16].

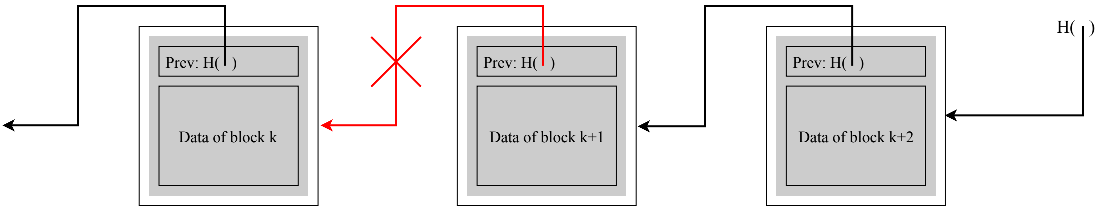


Figure 2.3: If block *k* is modified, the hash pointer in block *k+1* will be incorrect.

2.1.4 Merkle Trees

A Merkle tree is a binary tree that is built on hash pointers instead of regular pointers. The data blocks are grouped into pairs of two and make up the leaves of the tree. For each pair, a node containing two hash pointers, one for each block of the pair, is created. These nodes constitute the next level of the tree. In a further step, these nodes are also grouped into pairs of two and for each pair a node that contains also two hash pointers, one for each child, is created. This procedure is continued until a single node is left. This node is the root of the tree. Figure 2.4 shows an example of a Merkle tree [NBF+16].

Due to the utilization of hash pointers, any alteration of a node or a leave can be detected, as long as the root hash is stored in a secure place. The tamper-evident property is

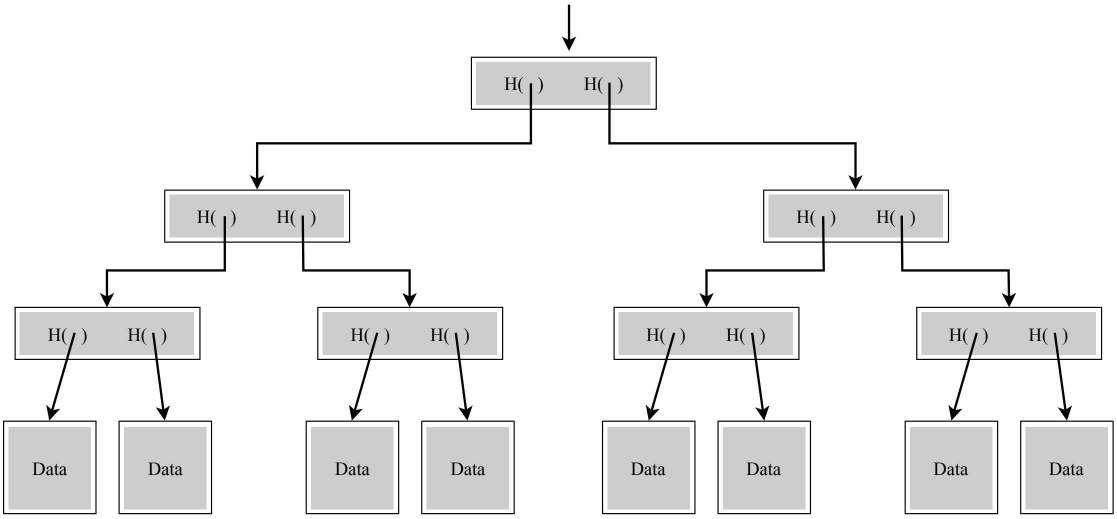


Figure 2.4: An example of a Merkle tree.

achieved in the same way as in the blockchain. Analogous to the blockchain, it is sufficient to remember only the hash pointer at the top in order to detect changes in the tree [NBF+16].

Proof of Membership

Unlike a blockchain, a Merkle tree allows a concise proof of membership, i.e., a proof that a certain data block is a member of the Merkle tree. As stated above, only the root hash has to be stored. If someone needs to show the membership of a certain block, it is sufficient to provide the data block and the nodes on the path from the data block up to the root. All other nodes that are not on this path can be ignored, since the nodes on the provided path are enough to verify the hashes all the way up to the root of the tree. Thus, it becomes possible to verify the membership of a particular data block in logarithmic time [NBF+16].

2.1.5 Radix Trees

A radix tree or prefix tree is an ordered tree data structure that is optimized for searching and is typically used to store dynamic sets or associative arrays. In a radix tree, a key represents the path to reach the value associated with that key. Figure 2.5 shows a radix tree that is constructed from the data set listed in Table 2.1. In order to avoid many nodes with null values, the branch of *house* and *houses* is degraded. If the value of a particular key should be retrieved, the path represented by the key determines the node in the tree that stores the corresponding value. For instance, if the value of the key *dodo* is needed, it is sufficient to start from the root and keep descending the entire path. The final result is the node with value *4* [Tạ18a].

Table 2.1: Data set that is used to construct the radix tree illustrated in Figure 2.5.

Key	Value
do	0
dog	1
dax	2
dogu	3
dodo	4
house	5
houses	6

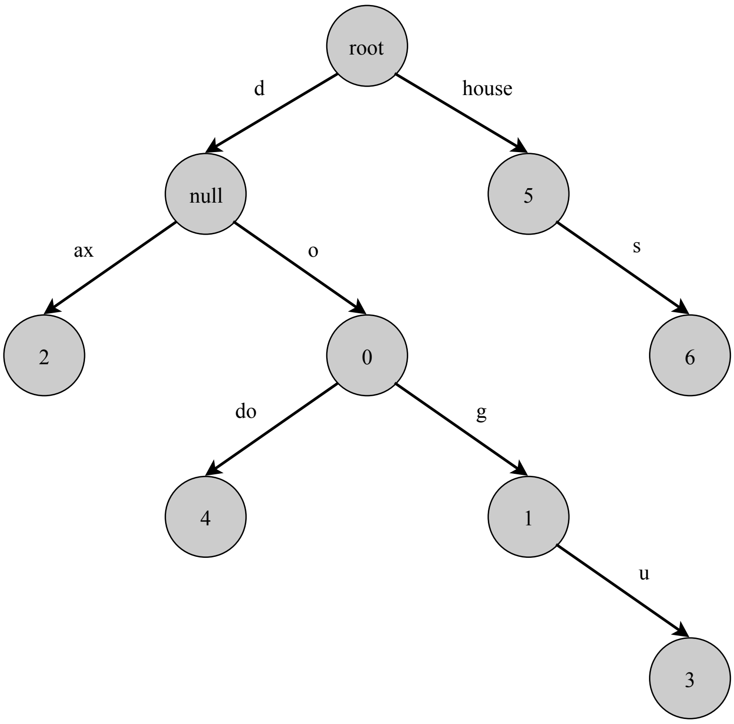


Figure 2.5: An example of a radix tree that is constructed from the data set listed in Table 2.1.

2.2 Basic Concepts of Bitcoin

As mentioned in Section 1.1, Bitcoin is the most prominent cryptocurrency today. First attempts in the area of blockchain interoperability have been made between Bitcoin and other Bitcoin-derived blockchains (e.g., Litecoin). Thus, this section gives a brief introduction to basic concepts of Bitcoin in order to understand protocols and approaches described in Chapter 3.

Bitcoin's blockchain belongs to the first generation of blockchains. It is mainly used to store payment transactions in a distributed and decentralized way [ZJ18]. With the stack-based scripting language Script, there can be realized some powerful applications such as efficient micropayments and lock times. But Script is not Turing-complete which leads to the consequence that people do not have the possibility to build arbitrarily powerful functions. Script was designed to be a simple and compact language that offers native support for cryptographic operations such as instructions for calculating hash values and for computing and verifying signatures. [NBF+16].

Bitcoin addresses

Bitcoin's identity management is completely decentralized. Users can generate as many identities as they want, and there is no central authority involved. Bitcoin uses a digital signature scheme known as the Elliptic Curve Digital Signature Algorithm (ECDSA). The scheme uses a key pair that consists of a private key and a public key. The idea behind addresses in Bitcoin is to equate the public key to the identity of a person or an actor. This allows every user to create a new identity by generating a fresh key pair. A Bitcoin address is derived from the public key. The public key is hashed with SHA-256 first and RIPEMD-160 subsequently. In a further step, a version number is prepended and a checksum is appended for error detection. Finally, the result gets base58-encoded in order to eliminate ambiguous characters. The purpose of these steps is to shorten and obfuscate public keys [NBF+16, TS16].

Bitcoin Blocks

As described above, a blockchain is a linked list that is composed of a series of blocks. Figure 2.6 shows a simplified structure of the Bitcoin blockchain and its blocks.

In Bitcoin, transactions are grouped into blocks and arranged in a Merkle tree. This grouping is an optimization, since miners have to reach consensus on each block and not on each transaction individually [NBF+16]. Furthermore, a chain of blocks is much shorter than a chain of transactions would be. Besides a set of transactions, a block contains a header that includes the following meta data [bit17]:

- **version:** The block version number specifies the block validation rules to follow.
- **previousBlockHeaderHash:** This value is a SHA-256 hash of the previous block's header.
- **merkleRootHash:** This value is a SHA-256 hash of the Merkle root.
- **time:** The block time specifies when the miner started hashing the header and is a Unix epoch time.
- **nBits:** This value defines the difficulty of the mining puzzle by specifying the encoded version of target threshold this block header's hash must be less than or equal to.

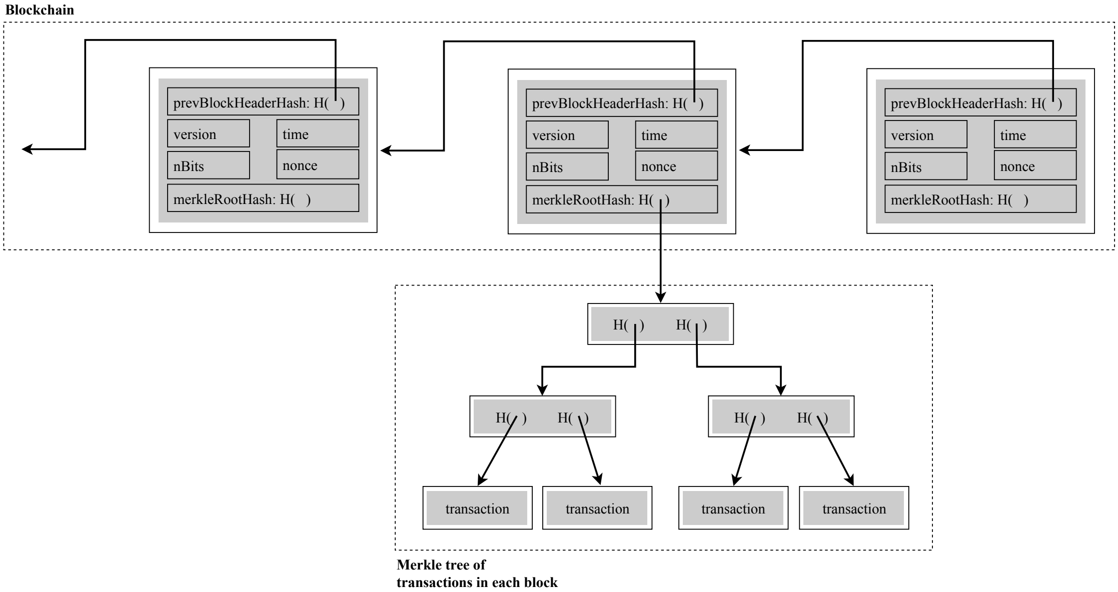


Figure 2.6: A simplified structure of the Bitcoin blockchain and its blocks.

- **nonce:** The nonce is an arbitrary number that is changed by the miners to modify the block header's hash in a way such that the produced hash is less than or equal to the target threshold.

In case transactions in the Merkle tree are modified, the hashes in higher level nodes are not going to match up. Therefore, a single change of a transaction has to be propagated to the root node of the Merkle tree. Since the Merkle root hash is included in the block header, the header's hash is not going to match up. Further, the previous block header hash included in the succeeding block is not going to match up. Thus, any change in the Merkle tree of each block or any modification of a block's header can be detected. In summary, the Bitcoin blockchain combines two data structures: A blockchain that links the blocks to one another and a Merkle tree that is internal to each block and arranges the block's transactions [NBF$^+$16].

Besides the tamper-evident property, the combination of these two data structures provides another useful feature. It is possible to verify payments without running a full node, i.e., a node that stores the entire blockchain. For the verification of payments, it is sufficient to keep only a copy of the block headers of the longest chain. As described above, a block header contains the Merkle root hash. In order to verify the membership of a particular transaction in the block's Merkle tree, only the branch of the Merkle tree starting at the root node and ending at the data node is required. All hash values of this branch are calculated and at the end the hash value of the branch's root node is compared with the block's Merkle root hash. If the hash values are equal, the transaction is a member of the block's Merkle tree. Thus, it can be verified that a particular transaction

Table 2.2: An example of transactions in a transaction-based ledger [NBF+16].

1	Inputs: $\emptyset$ Outputs: 25.0 → Alice	
2	Inputs: 1[0] Outputs: 17.0 → Bob, 8.0 → Alice	SIGNED (Alice)
3	Inputs: 2[0] Outputs: 8.0 → Carol, 9.0 → Bob	SIGNED (Bob)
4	Inputs: 2[1] Outputs: 6.0 → David, 2.0 → Alice	SIGNED (Alice)

is included in a block by simply comparing a single branch of the Merkle tree against the block header's Merkle root hash [NBF+16]. The availability of all block headers enables to verify that a particular block has enough confirmations in the blockchain. This verification mechanism is called Simplified Payment Verification (SPV) [Nak08].

Bitcoin Transactions

Bitcoin does not support an account-based system, because someone would have to keep track of all account balances. Instead, the distributed ledger keeps track of transactions. Transactions specify inputs and outputs. If a user wants to transfer coins to another, a transaction is created. The new transaction refers to at least one unspent transaction as input. An exception are transactions that generate new coins, these transactions do not reference any input transaction. Input transactions specify where the coins come from, since there are no account balances in Bitcoin. The outputs of a transactions specify the receivers of the coins. In order to authorize a transaction, the sender must sign it. Table 2.2 illustrates this concept by an example [NBF+16].

Transaction 1 does not reference any input transactions, because this transaction creates new coins. It has an output of 25 coins that are going to Alice. Since this transaction creates new coins, no signature is required. In Bitcoin, transactions that create new coins are called coinbase transactions and are used to reward the miners for successfully mining a block. In order to send some of the coins of transaction 1 to Bob, Alice creates transaction 2 that refers to transaction 1 as input. Alice refers to output 0 of transaction 1 (outputs are indexed beginning with 0) which assigned 25 coins to Alice. Furthermore, Alice must specify the outputs of the new transaction. She specifies two outputs, 17 coins to Bob and eight coins to Alice. In Bitcoin, all or none of a referenced transaction's output must be consumed. Thus, Alice creates the second output, where eight bitcoins are sent back herself. In order to authorize the transaction, Alice signs the entire content of the transaction [NBF+16].

It is also possible to create a transaction with multiple inputs. Assuming Alice and Bob want to pay David. They can create a single transactions with two inputs and one output. Due to the involvement of inputs that are owned by two different users, the transaction has to be signed by both Alice and Bob. This type of transactions is also called a 2-out-of-2 multi-signature transaction, since the transaction requires two out of two possible signatures to become valid [NBF+16].

Outputs of transactions can only be used as input in new transactions if they are unspent, i.e., they have not already been used as inputs in other transactions. Such outputs are referred to as Unspent Transaction Outputs (UTXOs) [TS16].

Listing 2.1 shows a low-level representation of a Bitcoin transaction.

Listing 2.1: The low-level representation of a Bitcoin transaction [NBF+16].

```
{
  "hash":
   "5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b"

  "ver": 1,
  "vin_sz": 2,
  "vout_sz": 1,
  "lock_time": 0,
  "size": 404,
  "in": [
    {
      "prev_out": {
      "hash":
   "3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
      "n": 0
      },
      "scriptSig": "30440..."
    },
    {
      "prev_out": {
      "hash":
   "7508e6ab259b4df0fd5147bab0c949d81473db4518f81afc5c3f52f91ff6b34e",
      "n": 0
      },
      "scriptSig": "3f3a4..."
    }
  ],
  "out": [
    {
      "value": "10.12287097",
      "scriptPubKey": "OP_DUP OP_HASH160
   69e02e18b5705a05dd6b28ed517716c894b3d42e
                          OP_EQUALVERIFY OP_CHECKSIG"
    }
  ]
}
```

The metadata section (line numbers 2-7 in Listing 2.1) includes the hash of the entire transaction that serves as a unique ID, a version number, the number of inputs, the number of outputs and the size of the transaction in bytes. In the example illustrated in Listing 2.1, the input array references two previous transactions (line numbers 8-23). The field *n* specifies the index of a transaction's output. *scriptSig* contains a script written in Bitcoin's scripting language Script and provides the signature that is needed to claim the transaction's output. Furthermore, the transaction has one output (line numbers 24-30). An output contains two fields. The *value* specifies the number of coins that should be spent to this output and *scriptPubKey* specifies a script that dictates under which conditions the output's coins can be redeemed [NBF+16]. Bitcoin scripts are discussed in the next section.

Bitcoin Scripts

Transaction outputs in Bitcoin do not just specify a public key. They actually specify a script. The most common case is to redeem a transaction output by providing the correct signature. In other words, a transaction output can be redeemed by a signature from the owner of address *X*. As outlined above, a Bitcoin address is a hash of the public key. Therefore, merely specifying the address *X* does not reveal the corresponding public key and it does not provide a way to verify the signature. Thus, a transaction's output must state that it can be redeemed by providing a public key that hashes to *X*, along with a signature from the owner of this public key [NBF+16].

Not only transaction outputs specify a script. Inputs contain also scripts instead of signatures. To confirm that a transaction redeems the coins of a previous transaction output correctly, the new transaction's input script is combined with the earlier transaction's output script. These two scripts are called *scriptPubKey* and *scriptSig.* They are simply concatenated by appending the *scriptPubKey* of the referenced transaction output to the *scriptSig* of the redeeming transaction, and the result must run without errors for the transaction to be valid. Examples of *scriptPubKey* and *scriptSig* are shown in Listing 2.1 [NBF+16].

Bitcoin's scripting language Script is stack-based. It was built specifically for Bitcoin and has many similarities with the stack-based programming language Forth. An important aspect is that Script is not Turing-complete. Table 2.3 shows common Script instructions and their functions [NBF+16].

Mining

Miners are connected to the Bitcoin network and listen for incoming transactions. Incoming transactions are validated, i.e., their signatures are verified and it is checked that the referenced transaction outputs have not been already spent. Miners maintain the blockchain. They listen for new blocks and verify them by checking all included transactions to be valid and whether there is a correct *nonce* in the block's header. Furthermore, miners build also their own blocks by grouping received and valid transactions to a block

Table 2.3: Common Script instructions and their functions [NBF+16].

Instruction	Function
OP_DUP	Duplicates the top item on the stack.
OP_HASH160	Hashes twice: first using SHA-256 and then a different hash function called RIPEMD-160.
OP_EQUALVERIFY	Returns true if the inputs are equal. Returns false and marks the transaction as invalid if they are unequal.
OP_CHECKSIG	Checks that the input signature is valid using the input public key for the hash of the current transaction.
OP_CHECKMULTISIG	Checks that the t signatures on the transaction are valid signatures from t of the specified public keys.

and by trying to find a nonce that makes the entire block valid. Finding a valid nonce requires the most work. Miners keep trying different nonces until the block's hash is below the target, i.e. the hash begins with the required number of zeros. In this step, the puzzle-friendliness property of a cryptographic hash function comes into play. It ensures that there is no solving strategy other than just trying random values for the nonce. This kind of consensus algorithm is called *proof-of-work*. Once a valid solution is found, the miner publishes the block to the network. Only if the other miners accept the new block, it is included permanently in the blockchain [NBF+16].

2.3 Basic Concepts of Ethereum

A further, very popular cryptocurrency is Ethereum. Its underlying blockchain technology belongs to the second generation of blockchains. Ethereum and Bitcoin have many similarities, but there are also significant differences between them. The main difference between Bitcoin and Ethereum is that Bitcoin is a currency and Ethereum is a computing platform that has a rich programming language. Further differences between these two systems are discussed in Section 2.4. In the following, fundamental principles of Ethereum and its underlying blockchain are highlighted.

2.3.1 Merkle Patricia Tree

The Merkle Patricia tree is a combination of the radix tree and the Merkle tree with some optimizations. It is the main data structure used in Ethereum, e.g., for storing transactions, the global state and an account's storage. In Ethereum, a tree node is stored as a key-value pair. The key is the hash of the node and the value is an array with 17 elements. The first 16 elements are indexed by a hex number from 0 to f, and contain hash pointers to descendant nodes. The 17th element stores the data of the node. This type of node is called *branch* [com14a, T18b]. Figure 2.7 illustrates the structure of a branch node in the Merkle Patricia tree. The key (i.e., the hash of the node) is

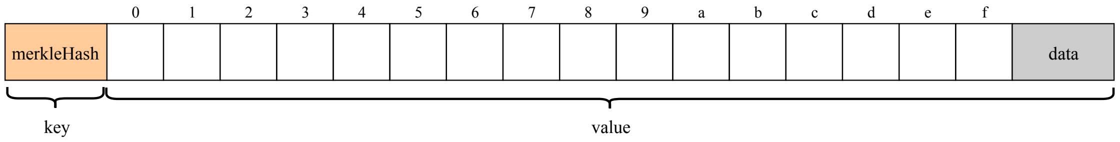


Figure 2.7: The structure of a branch node in the Merkle Patricia tree [T1̣8b].

Table 2.4: Data set that is used to construct the Merkle Patricia tree illustrated in Figure 2.8.

Path	Value
cab8	dog
cabe	cat
39	chicken
395	duck
56f0	horse

used to locate a node, whereas a path is used to find data in the tree by descending the entire path. In order to overcome inefficiency issues, two other types of nodes have been introduced in the Merkle Patricia tree. The first one is called *leaf* and the second one's name is *extension*. Both types of nodes have only two elements in their array. A leaf node stores a partial path in the first array element and the data in the second element. An extension node holds the partial path in the first array element just as the leaf node, but the second element contains the hash pointer to a descendant node. These two node types shrink the number of nodes with empty values in the tree and thus reduce also the number of steps that are needed to get the data associated with a particular path [com14a, T1̣8b]. Figure 2.8 shows an example of a Merkle Patricia Tree that is created from the data set listed in Table 2.4. The nodes *hashE*, *hashK* and *hashL* are leaf nodes, whereas the nodes *hashA* and *hashB* are extension nodes [T1̣8b].

Assuming the data located at path *395* is needed. In order to retrieve the correct data, the following steps have to be performed:

1. First, the path *395* is split into three parts *3*, *9* and *5*.
2. The root node can be found with *rootHash*, which is a hash pointer referencing the root node of the tree.
3. Since the first part of the path is *3*, the element at index *3* is retrieved. This element contains *hashA*.
4. In the next step, a lookup for *hashA* is performed in order to get the associated node. Furthermore, the value of the element indexed by *9* is retrieved. The value is *hashC*.

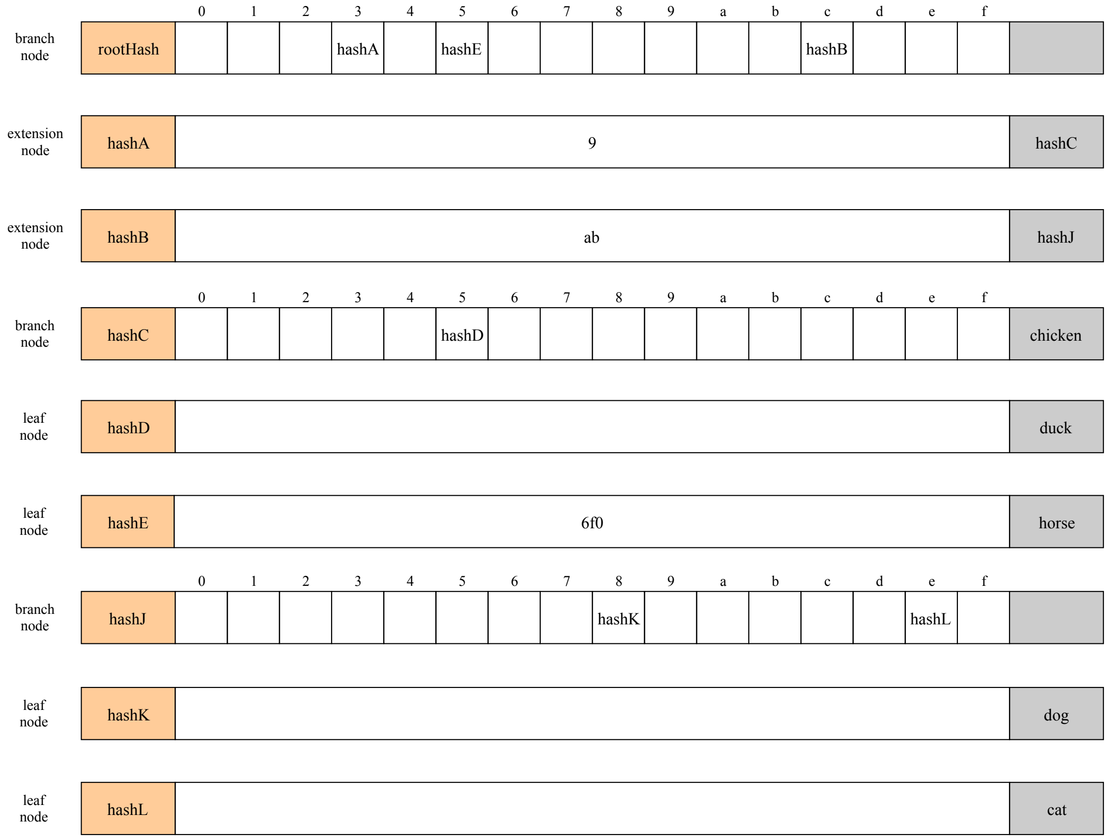


Figure 2.8: A Merkle Patricia tree that is constructed from the data set listed in Table 2.4.

5. A lookup for *hashC* leads to the next node. The value at array position *5* is *hashD*.
6. After the lookup of the node referenced by *hashD*, the value of the array's data element is retrieved. The result is *duck*.

An important aspect of Merkle Patricia trees is that they are fully deterministic, i.e., a Merkle Patricia tree with the same key-value bindings is guaranteed to be exactly the same down to the last byte and thus has the same root hash. Furthermore, Merkle Patricia trees enable insertions, lookups and deletions to be performed in logarithmic time [com16].

2.3.2 Ethereum Accounts

Unlike Bitcoin which uses the concept of UTXOs, Ethereum utilizes the account balance model, i.e., the Ethereum blockchain tracks how many coins each account has. When someone wants to spend coins, the system verifies the account's balance to make sure that the sender has enough coins before approving a transaction [Sun18]. In Ethereum, there are two types of accounts [com14b, Kas17, Ken18]:

- **Externally owned accounts**, that are controlled by private keys. These accounts have no code associated with them. Externally owned accounts can send messages to other externally owned accounts or to contract accounts. In order to send a message, a transaction is created and signed with the private key that controls the account.
- **Contract accounts** are controlled by their associated contract code. Unlike externally owned accounts, contract accounts cannot initiate new transactions on their own. They can only create a new transaction in response to another transaction they have received from another contract account or from an externally owned account. The contract's code can perform various actions, such as transfer tokens, write to internal storage, perform some calculation, create new contracts, etc.

Both account types store the following information [Woo14]:

- **nonce:** In case the account is an externally owned account, the nonce specifies the number of transactions that have been sent from the account's address. If the account is a contract account, the nonce represents the number of contracts created by this accounts.
- **balance:** The balance represents the number of Wei owned by this address. One Ether is equal to 10^{18} Wei.
- **storageRoot:** This field represents the hash of the root node of a Merkle Patricia tree that encodes the storage contents of the account and is empty by default.
- **codeHash:** This field specifies the hash of the Ethereum Virtual Machine (EVM) code of the account. In case the account is an externally owned account, the codeHash field contains the hash of the empty string.

As Bitcoin, Ethereum uses the ECDSA. Thus, a new identity can be created by generating a new key pair. Each account has an address that is constructed from the public key. An address is defined *"as the right most 160-bits of the Keccak hash of the corresponding ECDSA public key"* [Woo14].

2.3.3 Transactions and Messages

Ethereum is a *transaction-based state machine*, i.e., transactions that occur between different accounts cause the global state of Ethereum to move from one state to the next. Each action that occurs on the Ethereum blockchain is always triggered by a transaction fired from an externally owned account. A transaction is a signed data package that is created by an externally owned account, serialized and sent to the blockchain. A transaction represents either a message call or a contract creation which results in the creation of a new account with associated code [Kas17]. A message between two externally owned accounts is simply a value transfer. A message that is sent from an externally owned account to a contract account activates the contract account's code [Ken18]. Contracts can talk to other contracts by sending messages (also called internal transactions). In case a message is received, the associated code that exists on the recipient contract account is executed. The main difference between internal transactions and other transactions is that internal transactions are not generated by externally owned accounts. They are generated by contracts and are not serialized, i.e., they exist only in the Ethereum execution environment [com14b, Kas17].

A transaction contains [com14b, Kas17, Woo14]:

- **gasPrice:** This value specifies the number of Wei the sender is willing to pay per unit of gas. Gas is the fundamental unit of computation. Typically, a computational step costs one gas, but there exist operations that require higher amounts of gas because they are more computationally expensive.
- **gasLimit:** The gas limit represents the maximum amout of gas that should be used for the execution of the transaction.
- **recipient:** This value contains the address of the message call's recipient or, in case the transaction represents a contract creation, the field is empty.
- **value:** This value is equal to the number of Wei that should be transferred to the message call's recipient. In case the transaction is a contract creation transaction, this value is the starting balance of the contract account.
- **signature:** This information identifies the sender of the transaction.
- **init:** This field only exists for contract creation transactions and specifies the EVM code for the account initialization procedure. It returns the body of the account code that is permanently associated with the contract account.
- **data:** This field exists only for message call transactions and contains the input data (i.e., parameters) of the message call.

Since messages (internal transactions) are like transactions, except they are generated by contracts, they contain similar information such as the sender of the message, the recipient, the value to transfer, an optional data field and a gas limit [com14b].

2.3.4 Ethereum Blocks

As in Bitcoin, transactions are grouped into blocks that are published to the blockchain. This is done by the miners. The Ethereum blockchain has many similarities with the Bitcoin blockchain, but there are also differences. Unlike Bitcoin (which only stores a copy of the transaction list), Ethereum keeps track of both the transaction list and the entire, most recent state (e.g., account balances) [com14b]. Therefore, Ethereum blocks contain hashes referencing root nodes of Merkle Patricia trees instead of hashes that are pointing to root nodes of simple Merkle trees. In the following, some interesting header fields are described briefly [Kas17, Woo14]:

- **parentHash:** This value is the hash of the parent block's header.
- **ommersHash:** This field contains a hash of the block's list of ommers. An ommer is a block whose parent is equal to the current block's parent's parent. Due to the low block times in Ethereum (approximately 15 seconds) there are more competing blocks found by miners. The purpose of ommers is to reward miners for including these competing blocks. Ommer blocks are rewarded with a smaller amount than full blocks.
- **beneficiary:** beneficiary stores the address of the account that receives the fees for mining the block.
- **stateRoot:** This field stores the hash of the root node of the Merkle Patricia tree that is made up of the state after all transactions are executed.
- **transactionsRoot:** This field stores the root hash of the Merkle Patricia tree that contains all transactions of the block.
- **receiptsRoot:** This value is equal to the root hash of the Merkle Patricia tree that contains all receipts of all transactions listed in the block.
- **difficulty:** This field specifies the difficulty of the block and is used to enforce consistency in the time it takes to mine a block.
- **number:** This value is equal to the number of ancestor blocks connected in the blockchain and is increased for every new block.
- **gasLimit:** This field specifies the maximum number of gas expenditure per block.
- **gasUsed:** This value is equal to the number of gas used in transactions listed in the block.
- **mixHash** and **nonce:** These two values are combined and prove that a sufficient amount of computation has been carried out on the block.

At a first glance, storing the entire state in each block might be highly inefficient. Since the state is stored in a Merkle Patricia tree, only a small part of the tree needs to be changed after a block. In general, the vast majority of the tree should be the same between two adjacent blocks. To gain efficiency, the data are stored once and referenced multiple times using pointers (i.e., hashes of subtrees) [com14b].

2.3.5 Ethereum Virtual Machine

The Ethereum Virtual Machine (EVM) is the runtime environment for contracts in Ethereum and is completely isolated, i.e., the code running inside the EVM has no access to the network, filesystem or other processes [Com18c]. The EVM is a *quasi*-Turing-complete machine, since the computation is bounded through the gas limit. The virtual machine has a stack-based architecture, i.e., it uses a last-in, first-out stack to store temporary values [Kas17, Woo14]. Smart contracts that should be executed on the EVM must be encoded in EVM bytecode. Typically, programmers write their smart contracts in a higher-level language such as Solidity and compile their programs down to the EVM bytecode [Kas17].

Mining

Currently, Ethereum's mining process is almost the same as Bitcoin's. For each new block, miners have to solve a cryptographic puzzle. This method is called *proof-of-work*. To find a solution is very hard, but to verify an existing solution takes almost no time [com18b]. Ethereum developers plan to replace the proof-of-work algorithm with the proof-of-stake consensus algorithm, where a set of validators take turns at proposing and voting on the next block and the weight of each validator's vote is determined by the size of its deposit (i.e., stake) [com18b, Ros17].

2.4 Bitcoin vs. Ethereum

This section gives a brief overview of some important differences between Bitcoin and Ethereum. As stated above, Bitcoin is a decentralized currency, whereas Ethereum is a computing platform allowing users to build their own decentralized applications [Min17]. Bitcoin keeps only track of transactions, while Ethereum's blockchain stores both the transaction list and the entire, most recent state [com14b]. Unlike Bitcoin which uses the concept of UTXO, Ethereum utilizes the account balance model, i.e., the Ethereum blockchain tracks how many coins each account owns [com14b, Sun18]. In order to efficiently store the global state, Ethereum uses a modified version of the Merkle Patricia tree, whereas Bitcoin stores information in simple Merkle trees [com14b]. A further difference between Bitcoin and Ethereum is that unlike Ethereum's programming language, Script is not Turing-complete, i.e., it does not have the ability to compute arbitrarily powerful functions [Min17, NBF$^+$16]. Furthermore, the block time for Bitcoin is about ten minutes and for Ethereum 12 to 15 seconds [Min17].

CHAPTER 3

Related Work

This chapter describes related work and state-of-the-art methods in the area of blockchain interoperability and atomic cross-chain swaps. Section 3.1 presents the basic concept and limitations of the ACCS protocol proposed by TierNolan. Furthermore, different solutions focusing on the implementation of exchange and auction systems are outlined. Sections 3.2, 3.3, 3.4 and 3.5 summarize the concepts and ideas of the Lightning Network, relays, pegged sidechains and contributions regarding blockchain interoperability besides trading of cryptocurrencies. Section 3.6 concludes the chapter with a brief discussion about limitations of the related work.

3.1 Trading between Users of Different Cryptocurrencies

This section outlines contributions that focus on the exchange of coins between users of different cryptocurrencies, on real-time exchange services and on decentralized auction systems.

3.1.1 The Atomic Cross-chain Swap Protocol

First attempts in the area of blockchain interoperability have been made in 2013. A forum user with the pseudonym TierNolan proposed at `https://bitcointalk.org` the ACCS protocol that allows two or more users of different cryptocurrencies to swap their assets in an atomic and secure fashion [BJZ$^+$17, Tie13]. In its original version, the protocol can be used directly for trading between users of Bitcoin-derived blockchains [acc13]. Since the ACCS protocol has been used in other projects (e.g., BarterDEX) as well and detailed knowledge regarding the original version proposed by TierNolan is widely spread over the WWW (e.g., blogs, forum threads, etc.), this section addresses the protocol in depth in order to get a deeper understanding of the underlying concept.

The steps involved in the protocol are illustrated by an example, where Alice owns bitcoins, Bob owns litecoins and they want to swap a certain amount of coins based on the current exchange rates. Since there are two different blockchains involved, both parties need an address for both cryptocurrencies, such that Alice can send bitcoins to Bob's Bitcoin address and Bob can send litecoins to the Litecoin address of Alice. According to the protocol rules, Alice and Bob have to perform the following steps in order to redeem each other's funds [BJZ+17, Tie13, xHi15]:

1. Alice and Bob exchange their public keys.

2. Alice generates a random number x and computes the hash commitment $y = hash(x)$.

3. Alice creates a transaction Tx1 with a scriptPubKey which dictates that b bitcoins under the control of Alice can be redeemed

    - by supplying Bob's signature and a preimage z of y such that $y = hash(z)$ or
    - by supplying both the signature of Alice and the signature of Bob.

4. In order to get back possession of her b bitcoins in case of trade cancellation, Alice creates a *time-locked* (e.g., locked for 48 hours) transaction Tx2 that uses Tx1 as input transaction, and must be signed by Alice and Bob to claim the coins of Tx1 (i.e., the second condition of the scriptPubKey of Step 3 must be fulfilled). Therefore, Alice sends Tx2 to Bob, Bob signs it and returns the signed transaction to Alice. If Bob aborts the trade at any step, Alice can get back her spent coins by signing and publishing Tx2.

5. Alice broadcasts Tx1 to the network.

6. Bob creates a transaction Tx3 with a scriptPubKey which dictates that l litecoins under the control of Bob can be redeemed

    - by supplying the signature of Alice and a preimage z of y such that $y = hash(z)$ or
    - by supplying both the signature of Alice and the signature of Bob.

7. In order to get back possession of his l litecoins in case of trade cancellation, Bob creates a time-locked (e.g., locked for 24 hours) transaction Tx4 that uses Tx3 as input transaction, and must be signed by Alice and Bob to claim the coins of Tx3 (i.e., the second condition of the scriptPubKey of Step 6 must be fulfilled). Therefore, Bob sends Tx4 to Alice, Alice signs it and returns the signed transaction to Bob. If Alice aborts the trade at any step, Bob can get back his spent coins by signing and publishing Tx4.

8. If Tx1 has enough confirmations in the blockchain (i.e., the block that contains Tx1 is buried under enough blocks in order to become irreversible with high probability), Bob broadcasts Tx3 to the network.
9. If Tx3 has enough confirmations in the other blockchain, Alice creates and broadcasts a transaction Tx5 that redeems Bob's litecoins from transaction Tx3 by supplying the secret number x in the transaction's scriptSig.
10. Since Alice has revealed x in transaction Tx5, Bob can create a transaction Tx6 that redeems the bitcoins of Alice from transaction Tx1 by supplying x in the transaction's scriptSig.

The basic idea of the protocol is that Alice can only redeem Bob's litecoins by revealing the secret number x, thereby enabling Bob to use x in order to redeem the bitcoins of Alice on the other blockchain [BJZ$^+$17]. This lock is called *hashlock*, since the output script only transfers ownership of coins from one party to another if a preimage of the hash is provided. The second lock mechanism used in this protocol is called *timelock*. A timelock restricts the spending of some coins until a specified future time or block depth. This time limit ensures that a party should be able to get back possession of the spent coins in case the other party aborts the trade. In the example outlined above, the bitcoins of Alice should be locked for a longer time than Bob's litecoins. This should give Bob enough time to find out x and redeem the coins of Alice. A contract that utilizes both locking mechanisms is called Hashed Time-Locked Contract (HTLC) [BH17, htl16]. The concrete timelocks depend on the involved cryptocurrencies respectively blockchain technologies. For instance, many exchanges accept a Bitcoin transaction after it is buried under six blocks, whereas a Litecoin transaction is considered to be confirmed after it is buried under twelve blocks. Furthermore, the block creation rate of Litecoin is four times faster than in Bitcoin [BJZ$^+$17, TS16].

Further discussions regarding the ACCS protocol can be found in [acc16, ato16].

Properties of ACCS

According to [xHi15], the ACCS protocol should satisfy the following two properties:

- **Liveness**: *"There always eventually exists a step each party can do either to make progress within the protocol or to revert its effects."*
- **Safety**: *"The protocol never deadlocks and neither party can steal the other party's coins."*

xHire provides a proof in order to show the satisfaction of the previously mentioned properties [xHi15]. In this proof, both properties are shown separately. Deviations from the protocol might result in losing coins, but do not harm the other party [xHi15]. Furthermore, Bentov et al. provide a proof of security for the steps involved in the ACCS protocol [BJZ$^+$17].

Limitations of ACCS

According to [BJZ+17], it would require users to wait for many minutes or even hours until a trade is completed, since transactions involved in the trade need to be confirmed by a safe number of blocks. For instance, in the Bitcoin system a transaction needs to be buried under six blocks in order to become irreversible with high probability. This can take up to one hour.

Furthermore, ACCS does not enable participants to respond to price fluctuations in a rapid manner, such that users can alter their trading positions within seconds. Since ACCS does not support this kind of real-time exchange, price discovery, where traders observe alterations in the bid and ask orders on the exchange and thus modify their trading positions, is also not possible [BJZ+17].

Mercury

In 2015, a forum user with the pseudonym mappum announced at `https://bitcointalk.org` an early alpha preview of *"the world's first trustless cryptocurrency exchange"* [map15] service called Mercury. The Mercury Wallet implements the ACCS protocol. There are binaries for Windows, Mac OS X and Linux available. The alpha preview supports Bitcoin, Litecoin and Dogecoin. mappum stated that many more cryptocurrencies and assets will be added in the future if they are highly demanded by Mercury users. But Mercury became defunct because of missing popularity [BJZ+17]. The source code can be found at `https://github.com/mappum/mercury`.

3.1.2 Tesseract

Tesseract is a secure real-time cryptocurrency exchange service proposed by Bentov et al. [BJZ+17]. In order to achieve security and performance, Tesseract relies on a trusted execution environment (TEE), specifically Software Guard Extensions (SGX) developed by Intel. Intel SGX are a set of x86 hardware instructions enabling the execution of user programs in guarded, tamper-resistant memory areas. This memory areas are called enclaves. Enclaves are isolated from other processes, i.e., only the enclave can access its own memory region, any other attempt to access the enclave is blocked by the CPU. SGX do not need to rely on the security of operating systems or a cloud provider's hardware. Even if a malware has gained kernel privileges, enclaves are not vulnerable to attacks performed by such privileged software [SWG+17]. Furthermore, Intel SGX provides a mechanism called attestation. Attestation means that Intel SGX can confirm that an output represents the result of an execution. Remote users can verify that an attestation is correct [BJZ+17].

In Tesseract, the enclave code is hard-coded with the hash of Bitcoin's genesis block or any other checkpoint of the blockchain. On startup, the enclave loads the most recent block headers, each header is validated and added to a FIFO queue stored inside the enclave. For each cryptocurrency that is supported by Tesseract, the enclave will maintain such a queue. In a next step, a key pair (secret key, public key) for each supported cryptocurrency is created. Tesseract maintains for each cryptocurrency a deposit, therefore it will attest that the public key is the deposit address. The attestation of the public key should be published through different services like websites or even blockchains.

In order to open a Tesseract account, users first need to spend a *“significant enough amount”* into a deposit address of the exchange by creating a deposit transaction. The output script of this transaction dictates that after a specified time limit the user can get back control over the spent coins. Before the time limit is reached, only the enclave can redeem the coins by creating a signature with its secret key. After the deposit transaction has been buried under enough blocks in the blockchain, the user sends an evidence of the confirmed deposit to the enclave. If the deposit transaction is valid, Tesseract will credit the user’s deposited amount into her account entry in the array of users that is kept inside the enclave. Now, the user can trade in real-time with other Tesseract users by transmitting bid or ask orders to the Tesseract server. Tesseract matches bids and asks. User requests are recorded in the order book that is stored inside the enclave. Tesseract broadcasts an anonymized version of this book allowing anyone to observe the price spread. For each successful trade, Tesseract collects a proportional fee.

The Tesseract exchange service overcomes the limitations of ACCS outlined in Section 3.1.1. Bentov et al. provide a reference implementation that supports Bitcoin, Ethereum and similar cryptocurrencies. It is crucial to note that researchers at Austria’s Graz University of Technology developed a proof-of-concept that can extract RSA keys from SGX enclaves running on the same system within five minutes [SWG$^+$17].

3.1.3 Komodo’s BarterDEX

BarterDEX, currently (March 2018) in BETA phase of development, is a technology of the Komodo platform [kom18] enabling users to trade cryptocurrencies directly from one party to another without the need of a trusted third party. The Komodo platform focuses on providing end-to-end blockchain solutions for developers. Komodo’s objective is to offer blockchain solutions that can be customized to meet desired needs and are easy to deploy. The developers of Komodo focus on blockchain entrepreneurs and average cryptocurrency investors with the objective to form an economic ecosystem. The main parts of Komodo platform are

- BarterDEX, an atomic swap-powered decentralized exchange,
- Jumblr, an open-source and decentralized cryptocurrency anonymizer,
- Delayed Proof-of-Work (dPoW), a consensus algorithm to maintain the network and
- the decentralized initial coin offering.

BarterDEX is a decentralized auction system and combines three key components: order matching, trade clearing and liquidity provision. The first component is responsible for pairing an user's offer to buy with another user's offer to sell. Bids and offers placed by users on the network are recorded in a decentralized order book. Trade clearing is the process of swapping assets between the trading parties using a variation of the atomic cross-chain protocol proposed by TierNolan. *"In addition, the trading happens in real-time, through automation, on a decentralized peer-to-peer network, supporting a countless number of separate blockchain projects, while providing a speed and (eventually) liquidity comparable to that of a centralized exchange. BarterDEX also automatically calculates the appropriate mining and transaction fees for the blockchains involved"* [kom18]. In order to overcome the problem of low liquidity, BarterDEX creates Liquidity Provider Nodes. Liquidity Provider Nodes are trading parties that provide liquidity to the exchange by buying and selling assets. They gain their profit from the spread between bid and ask orders and bring price stability.

BarterDEX relies on the concept of UTXO that can be spent as an input in a new transaction.

As of February 2018, the public Komodo community has already performed 21,000 atomic swap trades on BarterDEX [kom18].

3.1.4 Republic Protocol

The Republic protocol [rep18] is a decentralized open-source dark pool exchange enabling users to trade assets across the Bitcoin and Ethereum blockchains through atomic swaps. A dark pool is a type of trading system that allows investors to place orders and make trades without publicly revealing their intentions during the search for a trading partner. Orders are recorded in a hidden order book and matched by an engine. Republic ensures that market-sensitive information such as price and volume at a certain position are not exposed to the public. Therefore, each order is broken down into a large number of order fragments that are distributed throughout the network. Orders can only be reconstructed by combining a majority of the order fragments. Republic utilizes a decentralized peer-to-peer network in order to initiate an atomic swap between two traders after a successful order match. Peer-to-peer nodes race to discover order matches. In order to incentivize participants to run order matching nodes, the Republic protocol introduced so-called REN tokens that are paid as order fees. Found matches are registered such that other nodes can recognize which orders have been closed. Furthermore, the associated traders are notified. This is done on the Ethereum network, acting as trusted third party that will always behave honestly (assumption on which Republic is built on) [ZW17].

3.1.5 Atomic Cross-chain Swap Protocol with Directed Graphs

Herlihy describes a protocol for atomic cross-chain swaps using HTLCs [Her18]. The author's approach relies on a strongly connected directed graph $G = (V, E)$, where each vertex $v \in V$ represents a party and each edge $(v, w) \in E$ represents an asset transfer from v to w. The hash-locked secrets are generated by so-called leaders. The set of leaders $L \subset V$ forms a feedback vertex set. L is a feedback vertex set for G if its deletion leaves the graph acyclic. Finding a minimal feedback vertex set for G is NP-complete [Her18]. However, there exists an efficient 2-approximation [BG96]. The author shows that the protocol does not satisfy necessary requirements if G is not strongly connected or if G is strongly connected but L is not a proper feedback vertex set.

For simplicity, the author assumes that the directed graph, that represents the swap, is constructed by a (possibly centralized) clearing service. Each trading party creates a secret with the corresponding hash-lock and sends the hash-lock along with its trading position to the clearing service. The service combines the received hash-locks and trading offers and creates the corresponding directed graph along with the set of leaders forming a feedback vertex set. Furthermore, the clearing service creates a vector of the leader's hash-locks and a global deadline. If not all contracts are triggered at or before the deadline, the parties' assets will be refunded. For each edge $(v, w) \in E$, a smart contract is created and initialized with the asset to be transferred, the involved parties v and w, a time-lock vector depending on the position of that edge in the graph and with the hash-lock vector. Assets are only transfered if no time-lock of the time-lock vector has expired and all secrets have been revealed. Starting at the leaders, instances of the smart contract are propagated through the swap graph in the direction of the edges. In contrast to the contract propagation, secrets are propagated in the opposite direction. Thus, a network of smart contracts is established in order to ensure that all requirements of an atomic cross-chain swap are satisfied [Her18].

3.1.6 Further Projects in the Area of Atomic Swaps

This section gives a brief overview of further projects experimenting with atomic swaps:

- **Lykke** [lyk18] is a semi-decentralized exchange for trading cryptocurrencies and fiat currencies.
- **Bisq** [bis18] is a fully decentralized exchange application utilizing a custom peer-to-peer network that relies on Tor.
- **Blocknet** [blo18] is a peer-to-peer protocol enabling the transfer of assets between nodes on different blockchains.
- **Decred** [dec17] community completed a cross-chain atomic swap between Decred and Litecoin in September 2017.
- **Altcoin.io** [alt17] is developing an atomic swap wallet focusing on bringing atomic swaps to the masses.

3.2 Atomic Swaps with the Lightning Network

The Lightning Network is a decentralized system for instant micropayments and was proposed by Joseph Poon and Thaddeus Dryja in 2016 [PD16]. It was designed to solve some technical limitations of the Bitcoin blockchain, especially the scalability problem. To reduce the load on the main Bitcoin blockchain, assets are transferred off-chain. Two parties open a so-called bidirectional payment channel that enables them to move assets in both directions without waiting for each transaction to be confirmed on the main blockchain. In order to open such a payment channel, they create a 2-out-of-2 multi-signature transaction on the blockchain and at least one of them spends coins into the 2-out-of-2 ledger entry. The coins can only be redeemed if both parties provide a signature. After this transaction has been confirmed on the blockchain, the parties can create signed transactions that spend the funds of the initial transaction. This transactions are exchanged using direct peer-to-peer communication. They are not broadcasted to the network and thus are not recorded by the ledger. After the parties have finished transacting with each other, the most recently exchanged transaction signature is broadcasted to the network in order to get the final balance recorded by the ledger. Thus, only the first transaction opening a payment channel and the last transaction representing the last balances of the participants are mined in the blockchain. All intermediate transactions happening in the payment channel are not recorded in the blockchain. Participants of the Lightning Network can also exchange off-chain transactions in case they are not connected directly via a payment channel. If no direct channel has been established between two participants, transactions can be exchanged via other participants in the network acting as nodes. In order to prevent intermediate nodes to steal funds of exchanged transactions, HTLCs are used [PD16, Sta16].

Speeding and scaling up the Bitcoin blockchain is not the only use case for which the Lightning network is applicable: In November 2017, Lightning Labs released a YouTube video showing an atomic cross-chain swap between Bitcoin and Litecoin [Tom17].

3.3 Relays

A relay is a smart contract that is interested in particular events occurring on another blockchain. For instance, a smart contract running on the Ethereum blockchain verifies Bitcoin transactions, thus enabling Ethereum accounts to receive Bitcoin payments. Relays are relying on the concept of SPV that is described in Section 2.2 [But16].

Relay contracts have been already implemented. BTCRelay [btc18] is a Bitcoin light-client in the form of a smart contract running on Ethereum. It enables Ethereum smart contracts to verify Bitcoin transactions. Another project is PeaceRelay [Luu17] that allows interactions between the two different Ethereum forks Ethereum and Ethereum Classic. In contrast to BTCRelay, relay contracts are running on both blockchains, i.e., PeaceRelay enables a two-way-relay.

Assuming Alice wants to exchange 50 ETH for one BTC, an atomic cross-chain swap can be established by a relay contract running on the Ethereum blockchain and specifying that whoever provides a proof that they sent one BTC to Alice's Bitcoin address X, can redeem the 50 ETH. The proof can be of the form of a Merkle proof of membership [But16].

If a blockchain's consensus algorithm works slowly, it takes a long time to verify for one blockchain that another blockchain achieved consensus on some operation. This limits the speed of cross-chain trades. In practice, where exchange rates may change rapidly, this limitation is a weakness for cross-chain swaps [But16].

3.4 Pegged Sidechains

Back et al. propose the concept of pegged sidechains that enable users to transfer assets between multiple blockchains. *"A sidechain is a blockchain that validates data from other blockchains."* [BCD$^+$14]. Furthermore, *"a pegged sidechain is a sidechain whose assets can be imported from and returned to other chains"* [BCD$^+$14]. An important property of the proposed technology is that assets, which are transferred between sidechains, can be moved back by the current owner. In order to avoid the need for nodes that maintain the target chain to track the sending chain, the asset transfers are performed by providing proofs of possession in the transferring transaction. The assets are locked on the first blockchain. In a further step, a transaction on the second blockchain is created. The transaction's input contains the cryptographic proof that the assets are correctly locked. This cryptographic proof is provided as SPV proof. While the assets on the first chain are locked, the assets can be freely moved within the second chain without further communication with the first chain. To move back the assets to the first chain, i.e., unlock the coins, the same principle is applied. The coins on the second blockchain are locked and a SPV proof is used in order to unlock the coins on the first chain. If the users of a blockchain are full validators of the other chain, a SPV proof has only to be provided in one direction, since the full validators are aware of the state of the other blockchain [BCD$^+$14].

3.5 Blockchain Interoperability besides Trading of Cryptocurrencies

In November 2017, a project with the name Crowd Machine has been unveiled. The software platform aims to allow programmers, application developers and even non-developers to build decentralized applications that run on a peer-to-peer network computer made up of mobile devices and PCs. This peer-to-peer network computer is referred to as the Crowd Computer. The decentralized applications are locked into a specific blockchain. Overall, the Crowd Machine consists of three components: The already mentioned Crowd Computer, the Crowd App Studio and the Crowd Share. The Crowd App Studio provides developers with the ability to model their application requirements as a diagrammatic presentation of logic, rather than writing code. This diagrammatic presentation is called a pattern. An application is formed by combining multiple patterns. Further features of the Crowd App Studio are a form design environment for the creation of user interfaces, a mechanism for integrating an application with external systems such as third party APIs or databases, and a built-in release management including a complete application lifecycle management functionality. The Crowd Share is a GitHub-like source repository, where developers can publish their code and are rewarded if their published code is used and executed by other applications. Applications created with the Crowd Machine can be migrated from one blockchain to another, if required [Bri18, Low17, Spr18].

Another project in the area of blockchain interoperability is Polkadot. Polkadot was founded by Gavon Wood, the co-founder of the Ethereum network. Similar to the intranet/internet synergy allowing open and closed networks to have trust-free access to each other, Polkadot focuses on a network of blockchains, *"where private and consortium chains can be firewalled from open and public chains like Ethereum without losing the ability to communicate with them on their own terms"* [wf17]. Polkadot's design aims to enable applications and smart contracts on one blockchain to communicate with applications on other chains [Woo17].

A further project that aims to connect multiple blockchains is the decentralized Cosmos Network. It is designed to allow various independent public and private blockchains to communicate and exchange value with one another. Further aspects addressed by Cosmos are scalability and a developer-friendly interface for building blockchains [KB18].

Block Collider is a *"high-speed distributed ledger built on sets of blocks from other blockchains, integrating those chains together and enabling many cross-chain features"* [ct17]. The basic idea of the Block Collider multi-chain is the integration of the current state of each connected blockchain into the multi-chain, i.e., the most recent Block Collider block references the most current block in each of the supported chains [ct17].

Another contribution worth to be mentioned is the Blocknet Protocol, a peer-to-peer protocol between nodes on different blockchains. The decentralized protocol enables cross-chain swaps and cross-chain data transfer in order to create a ecosystem for blockchain microservices [CM16, The18a].

3.6 Runtime Selection and Switchover

In the last years, many projects in the area of blockchain interoperability and especially atomic cross-chain swaps have been started. The discussed approaches focus on interoperability between predefined blockchains, i.e., on the asset transfer between two or more blockchains that are known in advance. To the best of our knowledge, there are no contributions in the field of runtime selection of blockchains, and there is no general solution that enables users to store data in a most-suitable blockchain that is selected at runtime based on certain criteria, such as storage costs and the average time needed for storing a piece of data. The suitability of a blockchain should be defined on the basis of certain blockchain metrics. Briefly summarized, the current state-of-the-art approaches do not integrate a mechanism for selecting the most beneficial blockchain at runtime and the switchover between selected blockchains in a single solution.

CHAPTER 4

Motivational Scenario

Due to the variety of blockchain applications, this section highlights a use case that will be used as reference for further analysis and evaluation of the proposed framework. As described in Section 1.1, there are various use cases for which blockchains are applicable. Cryptocurrencies such as Bitcoin, Litecoin and Namecoin have been the first popular systems that utilize the blockchain technology as decentralized bookkeeper in which all payment transactions are securely recorded. Besides holding of financial information, future applications of the blockchain technology are still a subject of research. Popular use cases that came up in the last years are digital voting, notary services, SCM, auditing, control of ownership rights, cloud storage and many more [Dou18]. A promising experiment utilizing the blockchain technology has been started in Brooklyn, New York. Dozens of solar-panel arrays spread across rowhouse rooftops are connected to a network, called the Brooklyn Microgrid. The project allows energy producers to sell excess electricity to other participants in the network. In 2017, the project had about 50 participants. The main focus of this experiment is to create a peer-to-peer energy trading system that is built on a blockchain and enables neighbors to trade energy among themselves. The possibility of energy sharing would allow participants to bypass traditional energy supply, and to sell their spare electricity, rather than give it away [Car18].

The motivational scenario outlined in the following addresses the collaboration between multiple business partners that aim to increase their competitive and collaborative advantage through the exchange of information. A key aspect of such an inter-organizational collaboration is the involvement of independent and even competing companies that do not trust each other. Figure 4.1 illustrates an example of information flows between multiple organizations. Inter-organizational Information Systems (IOISs) are automated systems that enable the flow of information between two or more enterprises. One popular example of an IOIS is an SCM system, where the sharing of information with suppliers and customers focuses on decreasing costs and on improving customer service [CPPRPM14, HF10, LSST06].

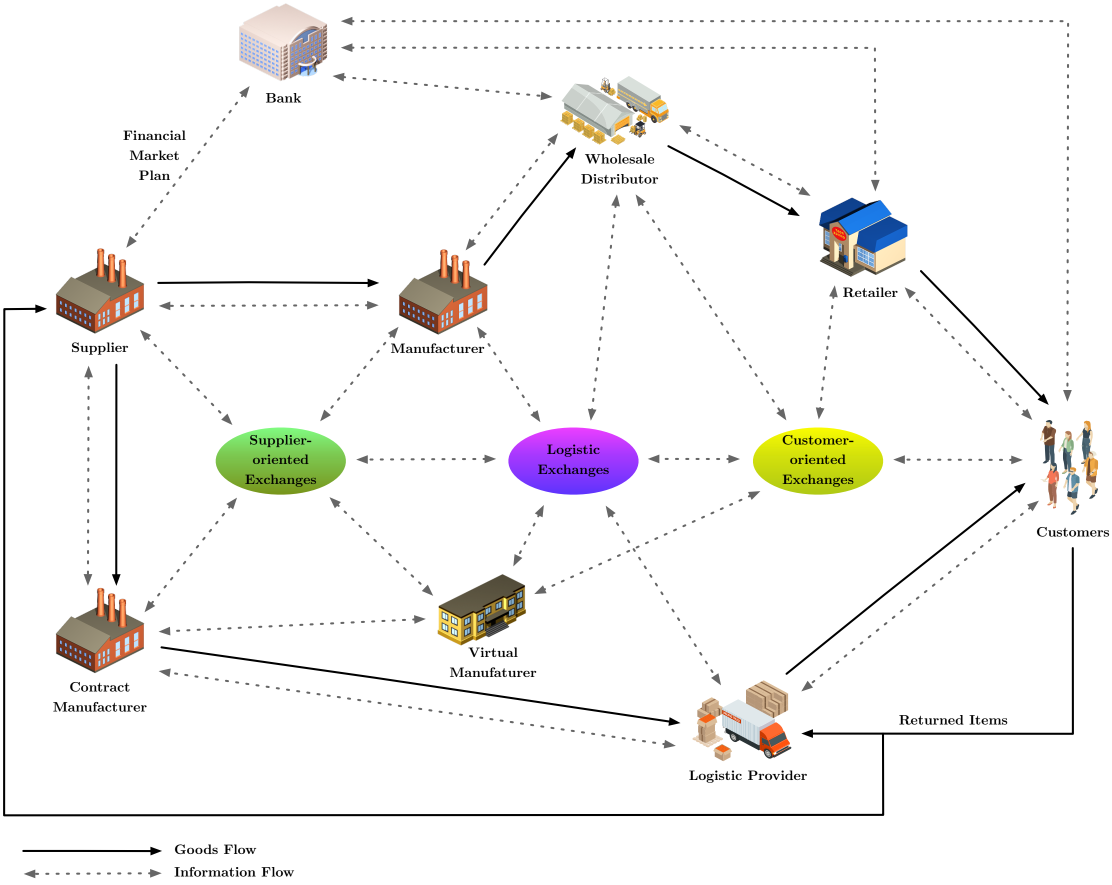


Figure 4.1: An example illustrating the information flows between multiple organizations [based on [kno11]]. The icons have been provided by vectorpocket (Freepik) and `Vecteezy.com`.

The growing awareness that exposing information to other companies can be beneficial to all involved parties led to an increased adoption of IOISs [LSST06]. In the last years, service-oriented computing gained much popularity in software development [HF10] [RD05]. Haki and Forte describe an inter-organizational information system architecture that utilizes the service-oriented approach [HF10]. A service-oriented architecture (SOA) is composed of loosely coupled and isolated services with limited dependencies on other shared resources, such as databases or APIs. It enables software developers to modularize complex systems by integrating services created and adopted by different vendors. One important advantage of SOA is that services can be consumed by different clients. Furthermore, different services composing an SOA can be developed with different languages and technologies. Two popular representatives for accessing web services are

SOAP and REST [SHG14].

Since various services developed and operated by different vendors can be involved in SOAs, quality and reliability of services become key aspects. Thus, Service Level Agreements (SLAs) are negotiated between service providers and service consumers [SBS18]. SLAs define functional and non-functional requirements, such as response time, data throughput or availability. In order to detect SLA violations, a monitoring approach can be implemented. A major difficulty in adopting a monitoring of services is that business partners have contradicting interests. For instance, a service consumer might be interested in revealing SLA violations in order to claim a penalty charge, while a service provider's objective might be the concealment of a violation [SBS18]. Schubert, Borkowski and Schulte proposed a solution for measuring and arbitrating SLA violations relying on a trusted third party [SBS18]. This approach does not require the involved parties to trust each other. The trusted third party component of their solution must be hosted by a neutral provider.

Similar to service invocations in SOAs, where different service providers and consumers are involved, transactions in cryptocurrencies take place between independent participants and must be secure enough to be considered indisputable. As mentioned in Section 1.1, Bitcoin and other cryptocurrencies are utilizing the blockchain technology as distributed ledger for securely recording payment transactions. The ledger replaces a trusted third party, and is operated by hundreds or thousands of miners distributed all over the world. Due to the fact that there is no central controlling entity and due to the tamper-evident property, a promising technology to get as a trusted third party for measuring SLA violations might be a blockchain. In an SOA, a blockchain acting as trusted third party can record various kinds of data and metrics, such as timestamps, memory usage and CPU utilization provided by both service consumers and service providers. The recorded data can be used for detecting SLA violations, for monitoring service invocation paths and thus serve as a basis for revealing performance bottlenecks, erroneous services, availability issues, insufficient throughput, etc.

Since the costs for storing data in a certain blockchain can increase in a rapid manner due to the sensitivity of cryptocurrencies for price fluctuations, the utilization of a predefined, static blockchain might not be a suitable approach. In order to overcome this limitation, the most beneficial blockchain should be selected at runtime based on certain blockchain criteria, such as average storage costs, and the average or median block time (i.e., the time it takes to mine a block). Furthermore, old data that is needed on the selected blockchain for further analysis or computation should be moved autonomously from the previously used blockchain to the currently selected chain.

To sum up, the motivational scenario focuses on an SOA that is made up of different services adopted and operated by several independent organizations. In order to ensure a certain level of service quality, the organizations agree on several service aspects such as availability, throughput, average time to answer, etc. in the form of an SLA. Due to the involvement of different and possibly competing organizations, the blockchain technology is used as bookkeeper that securely records various kinds of data, especially service quality aspects, published by the different service providers and consumers. If necessary, the blockchain compensates possible SLA violations, e.g., by collecting penalty charges through smart contracts. To keep the costs low, the most beneficial blockchain should be used as storage for the log data. In case another blockchain is more beneficial, this chain should be used as ledger for holding future log data. If old data located on the previously used blockchain is needed on the selected chain, the required data fragments should be moved towards this chain.

The challenge of this thesis and its proposed framework is to address the runtime selection of an appropriate blockchain and the movement of data between the former and another, more beneficial blockchain. The solution will not be restricted to the requirements of the described use case. The scenario outlined in this section merely serves as context in which the proposed framework will be analyzed and evaluated.

CHAPTER 5

Solution Approach

This chapter presents concrete requirements of the proposed framework followed by design decisions and implementation details. As mentioned above, the framework should monitor several blockchains. In case a blockchain is more beneficial than the chain that is currently used, a switchover should be initiated. The concrete criteria for the monitoring and the switchover are outlined in Section 5.1. Technical design decisions are discussed in Section 5.2. The chapter concludes with implementation details that are highlighted in Section 5.3.

5.1 Requirements

5.1.1 Monitoring

In order to select the most beneficial blockchain for the switchover, several metrics have to be gathered and analyzed by the framework for each supported blockchain. In the following section, metrics that are relevant for the runtime selection algorithm of the framework are described. Each metric is assigned to one of the following categories:

- **Costs:** This category includes all metrics that are used to observe the costs for interactions with a particular blockchain.

- **Security and Trust:** Metrics regarding trust in a particular blockchain, community consensus and controversies, security concerns and reputation belong to this category.

- **Performance:** This category covers all performance-related metrics, e.g., transaction throughput, inter-block time, etc.

Furthermore, each metric has an associated type:

- **Automatic:** Metrics of this type are automatically collected by the framework. There is no manual action or input required. Examples are transaction throughput, inter-block time, etc.

- **Predefined:** It is hard to measure metrics of this type automatically, since these metrics depend on the assessment of the user. Similar to approaches that acquire also consumer or user input for assessing and selecting service implementations in SOAs [BHMS05, WV07], the framework expects manual user inputs for quantifying metrics of this type. An example of a predefined metric is the reputation of a particular blockchain.

Figure 5.1 provides an overview of the metrics that are gathered by the proposed framework.

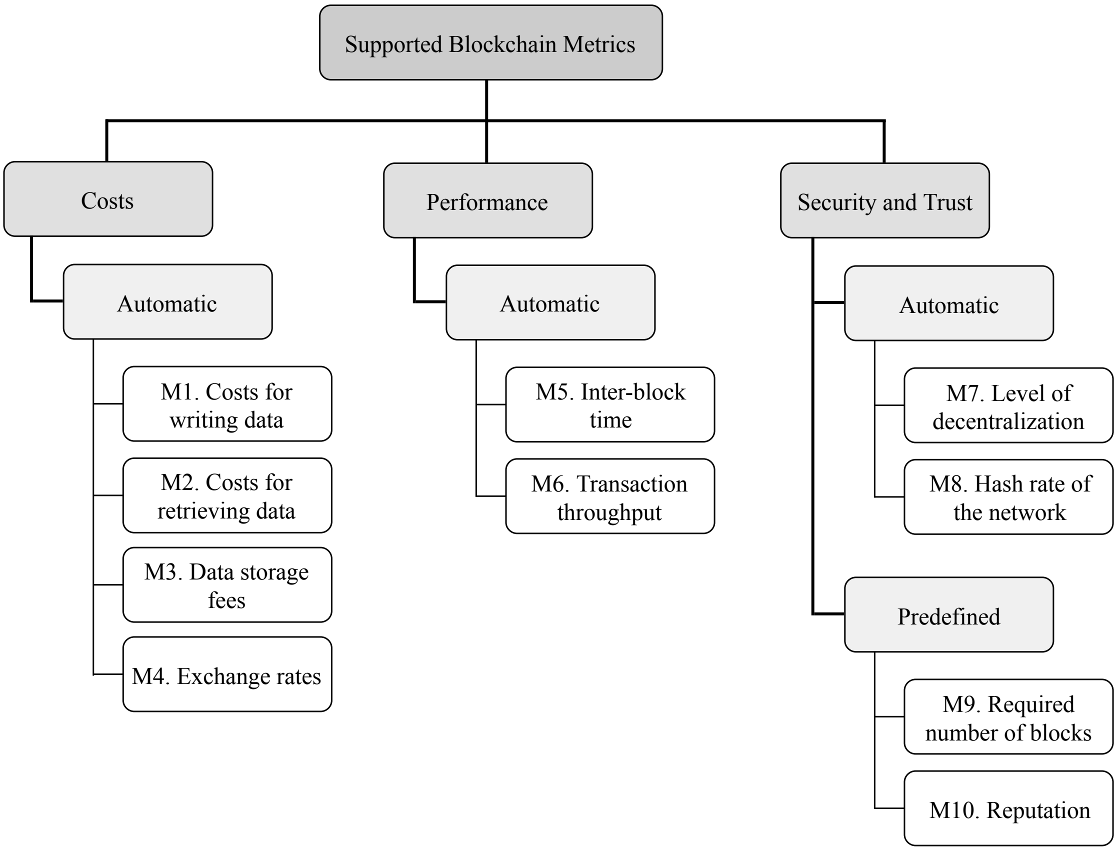

Supported Blockchain Metrics
Costs
Automatic
M1. Costs for writing data
M2. Costs for retrieving data
M3. Data storage fees
M4. Exchange rates
Performance
Automatic
M5. Inter-block time
M6. Transaction throughput
Security and Trust
Automatic
M7. Level of decentralization
M8. Hash rate of the network
Predefined
M9. Required number of blocks
M10. Reputation

Figure 5.1: Metrics that are gathered by the proposed framework.

In the following, each metric is described in detail:

M1. Costs for writing data into the blockchain

Description: This metric is an estimated value of the costs for writing one kilobyte of data into the blockchain. In some blockchains (e.g., Bitcoin), the storage costs might vary, since users can choose a priority for their submitted transactions. The priority affects the time until a transaction gets included in the blockchain and can be specified with transaction fees. In order to maximize their profit, miners include transactions with higher fees more likely than those with lower fees. Thus, transactions with higher fees will be included in the blockchain faster than those with lower fees. In order to address the interdependency between transaction fees and the time it takes until a transaction gets included in the blockchain, the framework user can specify custom fees. In case no custom fees are provided to the framework, the framework automatically estimates fees that will cause submitted transactions to get included within the next six blocks. The framework calculates the estimated storage costs on the basis of the provided or estimated transaction fees.

Category: Costs

Unit: EUR or USD per kilobyte

Type: Automatic

M2. Costs for retrieving data

Description: This value is an estimation of the costs for retrieving one kilobyte of data from the blockchain.

Category: Costs

Unit: EUR or USD per kilobyte

Type: Automatic

M3. Data storage fees

Description: The idea behind data storage fees is to charge users a fee to rent space in the blockchain. The longer a user wants some data records to be stored in the blockchain, the more expensive the charged fees will become. In March 2018, Vitalik Buterin suggested rental or storage fees for the Ethereum blockchain and outlined how such fees might work [Cos18].

Category: Costs

Unit: EUR or USD per kilobyte and hour

Type: Automatic

M4. Exchange rates

Description: This metric represents the current exchange rate between EUR or USD and the cryptocurrency of a particular blockchain. These values are useful for monitoring the growth of the exchange rates and for calculating costs that have to be paid for interactions with a particular blockchain.

Category: Costs

Unit: EUR or USD (e.g., the value of one bitcoin in EUR)

Type: Automatic

M5. Inter-block time

Description: The inter-block time specifies the rolling average of the time it takes to mine a block and is calculated from the blocks that have been mined during the last 24 hours. For instance, in Bitcoin a new block is mined approximately every ten minutes, whereas in Ethereum it takes 12 to 15 seconds until a new block is included in the blockchain [Min17]. The inter-block time is an important indicator of a blockchain's performance.

Category: Performance

Unit: Seconds

Type: Automatic

M6. Transaction throughput

Description: This value specifies the rolling average of the number of transactions that are processed per second by the system and is calculated from the transactions that have been mined during the last 24 hours.

Category: Performance

Unit: Number of transactions per second

Type: Automatic

M7. Level of decentralization

Description: This metric specifies the distribution of the network's hash power among miners, especially mining pools. The framework provides a mapping between miners and their proportion of mined blocks. The proportion is specified in percentage and calculated from the recent blocks that have been inserted into the blockchain during the last 24 hours. This mapping allows the user to make conclusions about the distribution of the network's hash power among miners. E.g., those miners that own or at least control a large proportion of the network's hash power can be easily identified.

Category: Security and Trust

Unit: For each address the proportion in percentage

Type: Automatic

M8. Hash rate of the network

Description: This metric specifies the estimated hash rate the network has performed in the recent 24 hours. The estimated hash rate is computed from the current difficulty and from the blocks that have been mined during the last 24 hours. The hash rates can be used to observe the growth of the overall network's hash power.

Category: Security and Trust

Unit: Hashes per second

Type: Automatic

M9. Required number of subsequent blocks

Description: This value specifies the number of blocks that should confirm a newly inserted block in the blockchain. The more blocks there are on top of a new block, the safer it is to assume that the new block is immutable and thus remains permanently in the blockchain. For instance, in the Bitcoin system a new block needs to be buried under six subsequent blocks in order to become irreversible with high probability [BJZ+17]. With an inter-block time of about ten minutes, this can take up to one hour.

Category: Security and Trust

Unit: Number of blocks

Type: Predefined

M10. Reputation

Description: This metric is a value between 0 and 10 and indicates the degree of reputation a blockchain is associated with. The reputation can include various properties such as trust, frequency of new feature releases, number of forks, community consensus and controversies, security concerns, etc. The value 0 indicates the worst reputation, whereas the value 10 represents an excellent reputation.

Category: Security and Trust

Unit: Positive integer between 0 and 10

Type: Predefined

The list of supported metrics is not restricted to the outlined metrics and can be extended, if desired.

5.1.2 Runtime Selection and Switchover

Based on the metrics that are gathered by the monitoring component of the framework, the runtime selection algorithm decides whether a switchover to another blockchain should be suggested. In general, a switchover should be suggested in the following two cases:

1. Too many metrics (the concrete number depends on the settings provided by the user) fall below specified thresholds or exceed them, i.e., they violate user-defined thresholds. In such a case, the most beneficial blockchain of the other supported chains is selected and a switchover is suggested, regardless of whether the currently used blockchain is the most beneficial chain.

2. The runtime selection algorithm detects a more beneficial blockchain and no metric has fallen below a threshold or has exceeded a specified limit.

Switchover Suggestions

On the basis of the provided settings and the gathered metrics, the framework automatically suggests the most beneficial blockchain. The user can subscribe to these switchover suggestions and define user-specific actions which are triggered for each new suggestion. For instance, the user can define to start a switchover to the suggested blockchain immediately after a switchover suggestion is received. Another possible action would be the suppression of subsequents suggestions.

Thresholds

For each gathered metric, an optional upper limit and an optional lower limit can be defined by the user. If a metric falls below the lower limit or exceeds the upper limit, the metric is considered as violated.

Metric-specific Timespans for Threshold Validations

Furthermore, the user can define a timespan for each metric. In case such a timespan is defined and a metric violates some thresholds, the metric is only considered as invalid if the particular metric has fallen below the lower limit or has exceeded the upper limit for the entire timespan. If no timespan is defined, a metric is considered as invalid immediately after a violation of a limit has been detected.

Switchover Condition for Threshold Validations

The framework allows users to define a condition that dictates whether a switchover should be suggested or not. This condition is evaluated on the basis of the threshold validation results of the supported metrics. A threshold validation result is a boolean value that indicates whether a metric is considered as valid or not. *True* means that the validation of a particular metric has been successful and no threshold has been violated. If a metric's value has fallen below or has exceeded user-defined limits for the entire timespan (if defined), the threshold validation result is false. Any new data input (e.g., a new block) will trigger a new threshold validation that computes fresh validation results for each metric. These validation results are applied to the specified switchover condition. Only if the condition evaluates to true, a switchover to another blockchain is suggested. The switchover condition can be any boolean expression composed of the threshold validation results. For example, a user can define that the condition evaluates to true and therefore suggests a switchover only if both the transaction throughput and the storage fees are considered as invalid (i.e., their threshold validation results evaluate to false). In the mentioned example, the condition is a simple logical conjunction of the negated threshold validation results and therefore only suggests a switchover if both threshold validation results evaluate to false.

Weighted Ranking

The framework calculates for each supported blockchain the benefit on the basis of the gathered metrics. For the calculation of a blockchain's benefit and the comparison of multiple blockchains, a weighted ranking is introduced. The user specifies for each metric a weight that indicates the degree of importance. Table 5.1 shows six weights that are offered by the framework. Each metric gets assigned one of five different scores by the framework. These scores are listed in Table 5.2. For each metric, the user has to provide an assignment that specifies which score is assigned to a metric on the basis of the metric's value. For instance, a user might define that inter-block times between 100 and 200 seconds are rewarded with a score of 2.

Table 5.1: Weights that are offered by the framework and their meaning.

Weight	**Meaning**
0	No importance
1	Very low importance
2	Low importance
3	Medium importance
4	High importance
5	Very high importance

Table 5.2: The score definitions.

Score	**Meaning**
0	Does not satisfy
1	Partly satisfies
2	Substantially satisfies
3	Almost satisfies
4	Fully satisfies

The framework performs the following computation steps for each supported blockchain in order to reveal the most beneficial chain:

- For each gathered metric, the metric's score is multiplied with the user-defined weight.
- The multiplication results (i.e., the weighted scores) are summed up.
- The blockchain with the highest weighted score is considered as the most beneficial chain and, if no metric is invalid due to threshold violations, is therefore selected for the switchover. In case two or more blockchains have the same score, one blockchain is selected in a random way. If the blockchain with the highest weighted score is not selected due to the violation of user-defined limits, the algorithm verifies the blockchain with the second highest score and so forth.

Table 5.3 shows an example of a weighted ranking, where two blockchains are evaluated. Blockchain A has a weighted score of 119, whereas Blockchain B has a weighted score of 124 and is therefore selected as destination chain for the switchover (under the assumption that no metric is invalid due to threshold violations).

Timespan for the Weighted Ranking

In case another blockchain is more beneficial, i.e., it has a higher weighted score in the ranking than the currently used chain, the user-defined timespan specifies the time

Table 5.3: An example of a weighted ranking, where two blockchains are evaluated.

		Blockchain A		**Blockchain B**	
Metric	**Weight**	**Score**	**Weighted Score**	**Score**	**Weighted Score**
M1	5	4	20	3	15
M2	3	4	12	4	12
M3	4	4	16	2	8
M4	2	2	4	4	8
M5	3	3	9	3	9
M6	3	2	6	3	9
M7	5	3	15	4	20
M8	4	3	12	2	8
M9	5	4	20	3	15
M10	5	1	5	4	20
Total	**39**	**30**	**119**	**32**	**124**

that has to elapse until a switchover is suggested. This timespan prevents immediate switchovers between blockchains. For instance, if the scores of multiple blockchains are very close together, it is possible that for one moment another blockchain is more beneficial and a few seconds later, a third chain is the preferred one. If no timespan is defined, the framework starts the switchover immediately after a more beneficial blockchain has been detected.

Amount of Data

The framework allows the user to define that the amount of data records which should be transferred from the currently used blockchain to the new one depends on the metric(s) that caused the switchover suggestion. For instance, if people are losing trust in the currently used blockchain, it might be essential to transfer all data or at least data of a specific period of time to the destination blockchain. Therefore, the framework will provide each metric's weighted score and threshold validation result in the returned switchover suggestions. On the basis of this information, the user can define a custom strategy specifying the period of time that determines the amount of data which should be moved towards the destination blockchain.

5.1.3 Resource Consumption

The framework should perform as efficiently as possible in terms of CPU utilization and memory consumption.

5.1.4 Supported Blockchains

Since Bitcoin and Ethereum are very popular blockchains, the proposed framework supports these blockchains by default. Furthermore, also Ethereum Classic and Expanse are integrated by default.

5.1.5 Extensibility

The list of supported blockchains is not static, i.e., new blockchains can be added later on, if desired. Adding new blockchains to the framework does not require to change the framework itself.

5.2 Technical Design

In the second part of this chapter, the technical design of the proposed framework is introduced. The technical design defines how the specified requirements can be solved by the means of concrete design and technology decisions. In the following sections, a general overview of the framework architecture, the most important design decisions, the technology stack, and concrete APIs between the involved components are described.

5.2.1 Architecture Overview

In Figure 5.2, the most important components of the framework's architecture are illustrated. The involved components are described in the subsequent sections.

External Data Sources

External data sources are components that provide relevant information for the framework but they are not part of the framework itself. The Blockchain Manager does not care about where the data come from, i.e., it is irrelevant if data is requested from public services, from public nodes or from a network node that runs on a private infrastructure, since the Metric Collector translates the received data into a neutral format that can be processed by the Blockchain Manager. We decide to run for each blockchain a private network node, since public services or public nodes have insufficient rate limits or can become temporarily unavailable in case too many other users call these services or nodes.

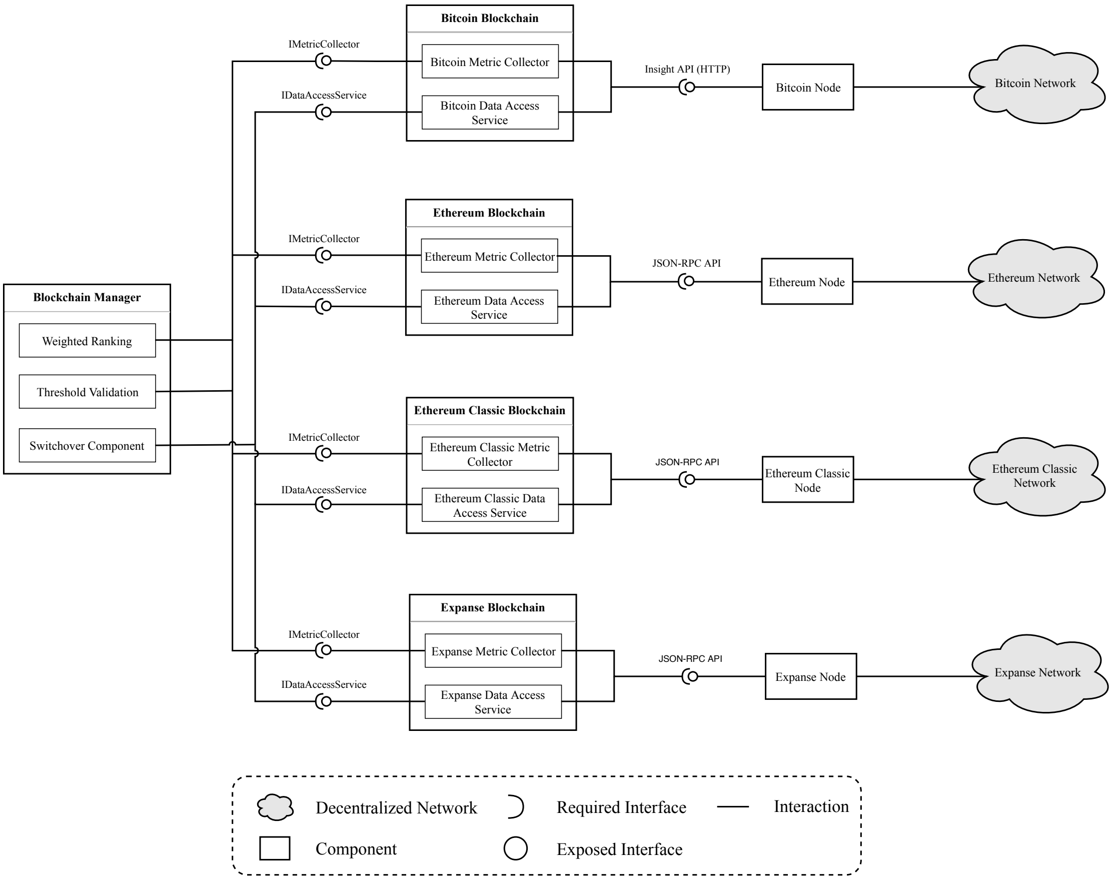


Figure 5.2: An overview of the framework's architecture.

The Blockchain Component

The Blockchain component represents a blockchain that has been registered at the Blockchain Manager. As illustrated in Figure 5.2, each registered blockchain contains the following subcomponents:

- Metric Collector
- Data Access Service

A blockchain's Metric Collector is responsible for downloading new blocks from the network, for calculating the values of the metrics M1 - M8 and for exposing these values through data outputs. The values of the metrics M9 and M10 are not computed by the Metric Collector, since these values must be provided by the framework user.

The design of the Metric Collector incorporates the reactive programming paradigm. In the reactive programming paradigm, data flows and the propagation of changes play a key role [Pat16]. If a data source changes its value, the change is propagated through the entire topology, i.e., each operator or observer that is part of the topology or is registered to receive notifications is informed about changes. Thus, reactive programming is well-suited for developing event-driven and interactive applications [BCC+13].

Each new block that is integrated into the blockchain affects the calculation of the metric values, and changes of metric values affect the weighted ranking system and the threshold validation mechanism of the Blockchain Manager. Therefore, every time a new block is received by the network node, the information is propagated through all computation steps that are internal to the Metric Collector and to the Blockchain Manager. For that reason, the reactive programming paradigm is an integral element of the technical design of both the Metric Collector and the Blockchain Manager. Figure 5.3 illustrates the design of the Metric Collector. The depicted stream topology is the same for all four Metric Collectors, merely some operators perform different calculations.

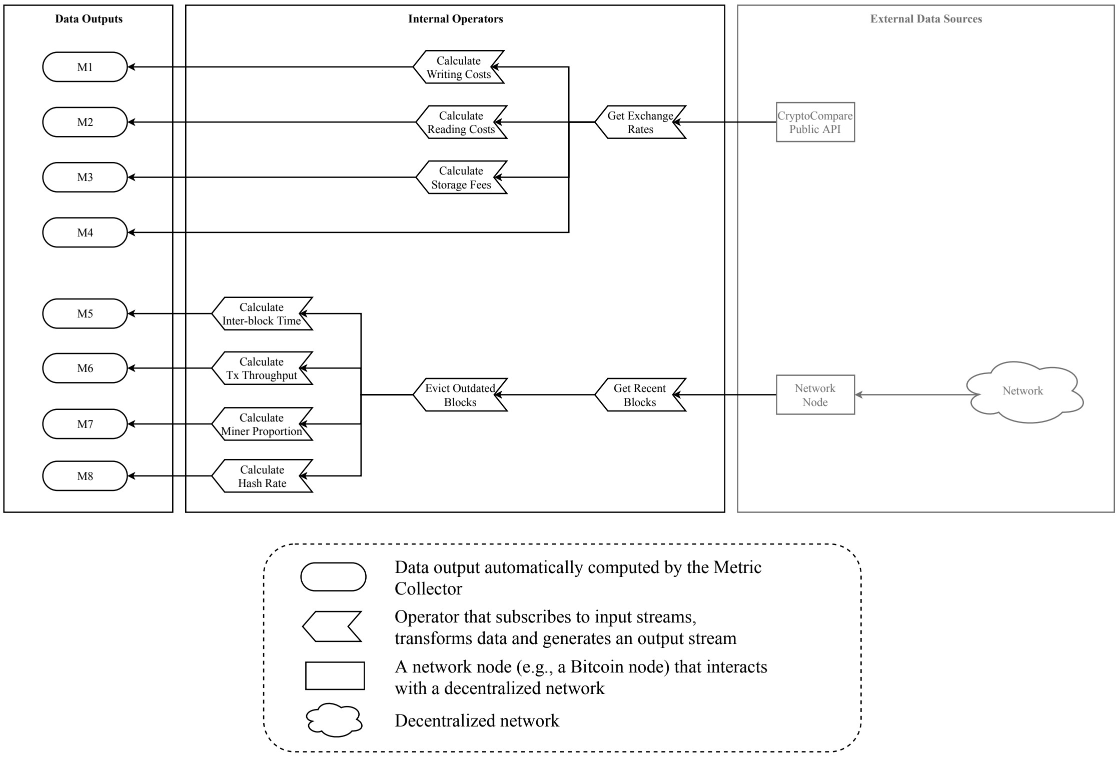


Figure 5.3: A simplified illustration of the stream topology that is utilized by the Metric Collector in order to calculate the metric values.

The operator *Get Exchange Rates* continuously requests the current price in EUR or USD for the cryptocurrency that is associated with the Metric Collector's blockchain. The exchange rates are propagated to the subsequent operators *Calculate Writing Costs*, *Calculate Reading Costs*, *Calculate Storage Fees* and to the data output *M4*.

The operator *Calculate Writing Costs* estimates the costs in EUR or USD that have to be paid for writing one KB of data into the Metric Collector's blockchain. A common practice to store data in the Bitcoin blockchain is to use the script operation code OP_RETURN. OP_RETURN marks a transaction output as invalid and accepts a user-defined sequence of up to 40 bytes [Apo17, Com18a]. In order to write data to the Bitcoin blockchain, a transaction with one input and two outputs (one output that spends the remaining coins and another one that holds the data) is sufficient. Such a transaction that stores 40 bytes of data in its second output has an overall size of 242 bytes. To store one KB of data, 26 transactions are required. The overall size of these 26 transaction is $25 \cdot 242 + 227 = 6,277$. 227 bytes is the size of the transaction that stores the remaining 24 bytes. The overall size of 6,227 bytes is multiplied with the user-defined transaction fees. If the user does not provide transaction fees, an estimation of transaction fees is requested from external services. Ethereum-based blockchains offer two possibilities for storing data into the blockchain. The first possibility is to deploy a smart contract that manages the data. Read and write operations are handled by the smart contract. The second possibility is to store the data in the input field of a transaction. Every transaction costs 21,000 gas, every non-zero byte that is stored in a transaction's input field costs 68 gas [Woo14]. The number of bytes that can be stored in a transaction is bounded by the current block gas limit. At the time of writing this thesis, more than one KB of data can be stored in an Ethereum transaction. Thus, only one transaction with a maximum gas usage of $21,000 + 68 \cdot 1024 = 90,632$ is required to store one KB of data in a transaction. Saving a 32-byte word to a smart contract's storage costs 20,000 gas [Woo14]. In case one KB of data should be stored in the contract's storage, $21,000 + 20,000 \cdot \frac{1024}{32} = 661,000$ gas has to be paid. 21,000 gas have to be paid for the transaction that contains the smart contract call. Therefore, storing data in transactions is the cheaper option. If the user does not set a preferred gas price, the Metric Collector requests the median gas price from the network node (Ethereum node, Ethereum Classic node or Expanse node). The result is propagated to the data output *M1*.

The operator *Calculate Reading Costs* computes the price in EUR or USD for retrieving one KB of data from the Metric Collector's blockchain. In the current design, no fees have to be paid for reading data located at Bitcoin, Ethereum, Ethereum Classic or Expanse. Thus, this operator returns always 0. This operator has been introduced if data is requested via smart contract calls that might cost some gas. The result is propagated to the data output *M2*.

At the time of writing this thesis, there are no rental fees that have to be paid for storing data in the Bitcoin, Ethereum, Ethereum Classic or Expanse blockchain. Since Vitalik Buterin proposed data storage fees for Ethereum, this metric is included in the Metric Collector's design in order to be on the safe side [Cos18]. The corresponding operator is *Calculate Storage Fees*. This operator returns always 0 and the result is propagated to the data output *M3*.

The operator *Get Recent Blocks* continuously receives new blocks from the network node depicted as rectangle in Figure 5.3. For interactions with the network node, the exposed interfaces are used. Detailed information about interfaces are described in Section 5.2.3. Each new block is propagated to the next operator *Evict Outdated Blocks.* This operator stores all received blocks in a list and continuously checks the creation dates of the blocks. If a block is older than 24 hours, it is removed from the list. Thus, the operator ensures that only those blocks that have been mined during the last 24 hours affect subsequent computation steps.

The operator *Calculate Inter-block Time* computes the average time (in seconds) between two blocks by applying the formula

$$\frac{24 \cdot 3600}{blockcount}. \tag{5.1}$$

The result is propagated to the data output *M5.* The same formula is used by the Metric Collectors of all supported blockchains (Bitcoin, Ethereum, Ethereum Classic and Expanse).

The transaction throughput is computed by the operator *Calculate Tx Throughput.* This operator applies the following formula in order to calculate the average number of transactions that are processed per seconds:

$$\frac{\sum_{i=1}^{n} txcount_i}{24 \cdot 3600}. \tag{5.2}$$

n denotes the total number of blocks and $txcount_i$ denotes the number of transactions of block i. The result is propagated to the data output *M6.*

The calculation of the distribution of the hash power (i.e., the calculation of metric M7) is performed in two different ways due to fundamental differences between Bitcoin and Ethereum-based blockchains. The operator *Calculate Miner Proportion* of Bitcoin's Metric Collector just needs to count the number of blocks for each miner that has mined at least one block during the last 24 hours. In a further step, each miner's number of blocks is divided by the overall number of blocks in order to get each miner's proportion. In order to compute the distribution of the hash power of the Ethereum, Ethereum Classic or Expanse network, uncle blocks must also be taken into account, since miners are also rewarded for the integration of these blocks. The result is propagated to the data output *M7.*

The operator *Calculate Hash Rate* computes the estimated number of hashes per seconds the network has performed in the last 24 hours. In order to calculate the overall hash rate of the Bitcoin network, the formula

$$\frac{blockcount}{144} \cdot \frac{D \cdot 2^{32}}{600} \tag{5.3}$$

is applied. *blockcount* denotes the number of blocks that have been mined during the last 24 hours and D specifies the network's current difficulty that must be taken into account by miners for solving the cryptographic puzzle. *144* is the number of blocks that are expected to get mined during 24 hours. $\frac{D \cdot 2^{32}}{600}$ specifies the expected number of hashes that have to be calculated in order to find a block with difficulty D. Every 2016 blocks the difficulty is changed such that the previous 2016 blocks should have been found at the rate of one every ten minutes (600 seconds) [Com17]. The hash rate of Ethereum-based networks is calculated by summing up the *difficulty* field of each block and each uncle block that have been included into the blockchain during the last 24 hours. The result is propagated to the data output *M8*.

The second subcomponent of the Blockchain component is the *Data Access Service*. The Data Access Service hides internal blockchain-specific implementation details and exposes interface methods for read and write operations. The Blockchain Manager stores data to or reads data from the blockchain through these interface methods. Thus, the Blockchain Manager does not need to care about blockchain-specific details.

The Blockchain Manager

The central component of the framework is the Blockchain Manager. The Blockchain Manager has a list of all supported blockchains. Each blockchain must be registered at the Blockchain Manager in order to get recognized by the selection and switchover algorithm. Furthermore, the Blockchain Manager exposes methods enabling the user

- to specify settings that affect the weighted ranking system and the threshold validation mechanism,
- to register a blockchain,
- to preselect a blockchain that is considered as the most beneficial chain for the startup phase,
- to write data to the current blockchain,
- to subscribe to switchover suggestions and
- to initiate a switchover to another blockchain.

The settings for the weighted ranking system include

- a defined weight for each metric,
- a score function for each metric that specifies which score is assigned to the metric based on its value (e.g., an inter-block time lower than 30 seconds results in a score of 4) and
- a switchover timespan that specifies the time that has to elapse until a new switchover is suggested.

Table 5.4: The data type of each metric.

Metric ID	Data type
M1	Positive Decimal incl. 0
M2	Positive Decimal incl. 0
M3	Positive Decimal incl. 0
M4	Positive Decimal incl. 0
M5	Positive Decimal incl. 0
M6	Positive Decimal incl. 0
M7	Mapping (key: String, value: Positive Decimal incl. 0)
M8	Positive Decimal incl. 0
M9	Positive Integer incl. 0
M10	Positive Integer ≥ 0 and ≤ 10

Each score function takes a metric value as input and returns an integer value between 0 and 5. The concrete data types of the metrics are outlined in Table 5.4.

The threshold validation settings contain

- for each metric a threshold validation function that takes the metric's value as input and returns true if a given value is valid or false, if the value violates at least one threshold,
- for each metric a timespan that defines how long a metric's value must fall below a lower limit or exceed an upper limit until the metric is considered as violated, and
- a switchover decision function.

The switchover decision function takes the latest threshold validation result of each metric and decides whether a switchover should be suggested. In case a switchover should be suggested, the function returns true, otherwise false. This function addresses the switchover condition for threshold validations introduced in Section 5.1.2. A threshold validation result of a particular metric is computed by applying the metric's value to the metric's threshold validation function with simultaneous consideration of the specified timespan. Only if the metric's threshold validation function continuously returns false and the timespan has elapsed, the threshold validation result is false. Each time, the threshold validation function of a metric evaluates to true, the counter for the elapsed timespan is reset.

The outlined configuration settings have to be provided by the framework user and they directly affect the internal logic of the Blockchain Manager. The internal structure of the Blockchain Manager is presented in Figure 5.4.

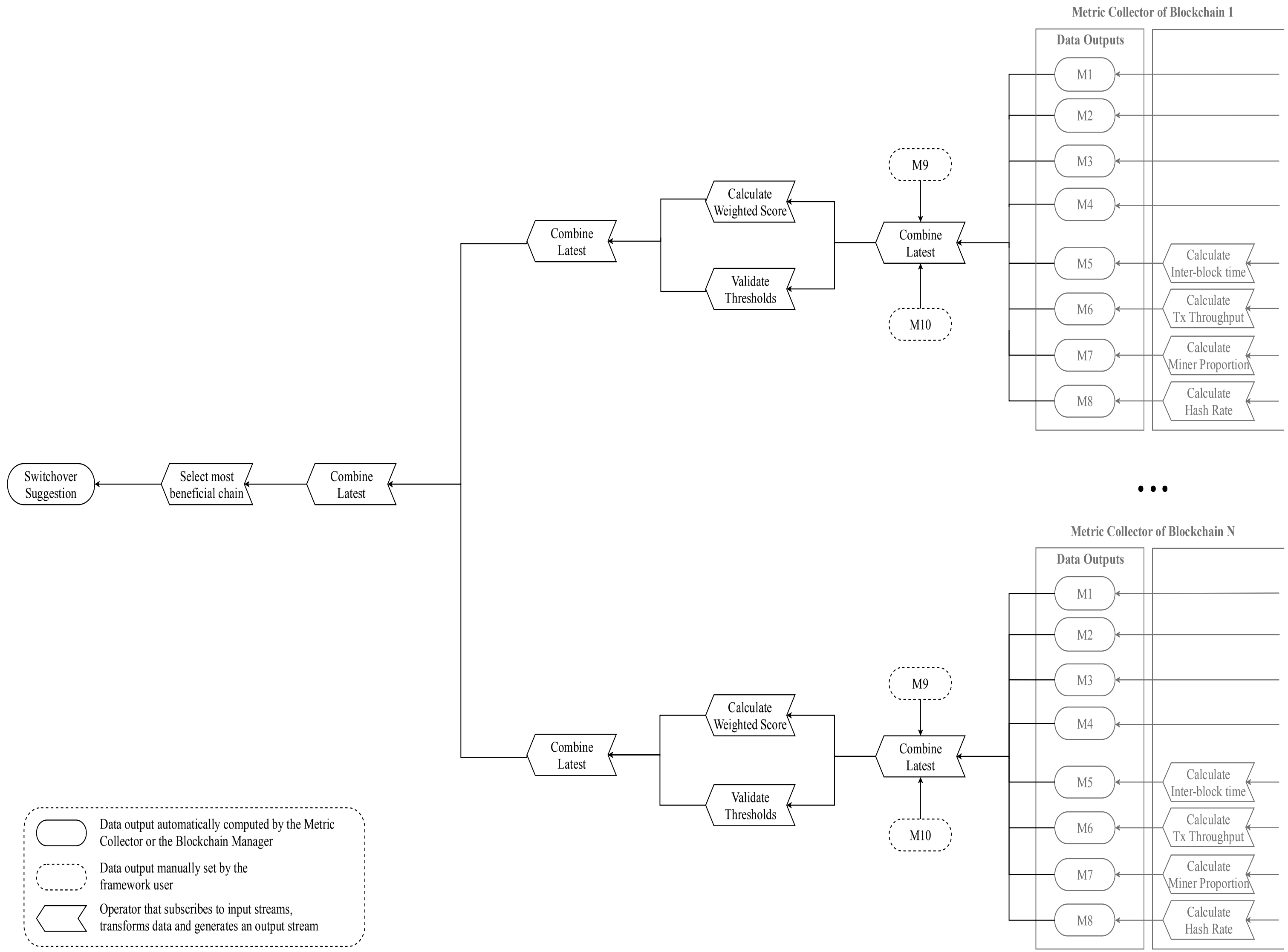

Metric Collector of Blockchain 1
Data Outputs
M1
M2
M3
M4
M5
M6
M7
M8
Calculate Inter-block time
Calculate Tx Throughput
Calculate Miner Proportion
Calculate Hash Rate
M9
M10
Combine Latest
Calculate Weighted Score
Validate Thresholds
Combine Latest
Metric Collector of Blockchain N
Data Outputs
M1
M2
M3
M4
M5
M6
M7
M8
Calculate Inter-block time
Calculate Tx Throughput
Calculate Miner Proportion
Calculate Hash Rate
M9
M10
Combine Latest
Calculate Weighted Score
Validate Thresholds
Combine Latest
Combine Latest
Select most beneficial chain
Switchover Suggestion
Data output automatically computed by the Metric Collector or the Blockchain Manager
Data output manually set by the framework user
Operator that subscribes to input streams, transforms data and generates an output stream

Figure 5.4: A simplified illustration of the internal stream topology that is utilized by the Blockchain Manager in order to determine the most beneficial chain.

The operator *Combine Latest* subscribes to all exposed data outputs of a Metric Collector and aggregates the latest value of each data output. Each time a new value arrives at any data output, the aggregation logic is performed. Thus, the emitted result of this operator reflects always the newest values of each data output and is propagated to the operators *Calculate Weighted Score* and *Validate Thresholds*. The operator *Combine Latest* is applied for each supported blockchain' Metric Collector as depicted in Figure 5.4.

The operator *Calculate Weighted Score* computes the weighted score for each metric as well as an overall blockchain score. In order to get each metric's score, the metric's score function provided with the weighted ranking settings is applied to the metric's value. In a further step, the metric scores a multiplied with the corresponding weights.

The operator *Validate Thresholds* applies each metric value to the metric's threshold validation function with simultaneous consideration of the specified timespan. Only if the threshold validation function continuously returns false during the entire timespan, the metric is considered as invalid. The operator's result contains for each metric the threshold validation result.

The results of the operators *Calculate Weighted Score* and *Validate Thresholds* are combined by a further application of the operator *Combine Latest* and the combined result is propagated to the next step which is also an application of the *Combine Latest* operator. This application combines the results of all registered blockchains and propagates the combination of all blockchain results to the operator *Select Most Beneficial Blockchain*. This operator evaluates the switchover decision function provided with the threshold validation settings for each blockchain's threshold validation results and selects those blockchain that has the highest weighted score and is considered as valid, i.e. the switchover decision function evaluates to false (no threshold violations).

The data output *Switchover Suggestion* enables the user to subscribe to switchover suggestions containing the current blockchain, the suggested blockchain and the weighted ranking and threshold validation results. The framework user can add custom logic to this subscription that determines on the basis of the switchover suggestion whether a switchover should be initiated and how many data records should be moved to the new chain.

In order to initiate a switchover, the exposed method for starting a switchover has to be called. This method accepts the new blockchain and a timespan (start date and end date) as inputs. On the basis of the timespan, the amount of data is selected that is moved to the new blockchain. During the switchover process, there are no restrictions regarding write operations. In case a switchover is still processing, all new data strings that should be stored into the current blockchain are buffered and written to the new blockchain after the switchover process has finished.

Due to the reactive nature of the framework design, any new data input is immediately propagated through the entire topology and the Blockchain Manager is able to react on new inputs by reevaluating the blockchains.

Table 5.5: The technology stack.

Technology	Usage
Java SE 10[1] (Oracle Java SE Development Kit 10)	Programming language
Apache Maven[2] 3.2.1	Build and dependency management
Spring Boot[3] 2.0.3.RELEASE	Consume HTTP services with Spring's RestTemplate
Jackson Datatype JSR310[4] 2.9.7	JSON serialization and deserialization
RxJava[5] 2.2.0	Building an asynchronous stream topology
web3j[6] 3.5.0	Communication with nodes on Ethereum blockchains
Logback-classic[7]	Logging
Apache Commons Codec[8] 1.11	Hex encoding and decoding
JUnit Jupiter[9] 5.1.0	Unit testing
mockito-core[10] 2.21.0	mocking framework for unit tests
gerba[11] 1.5.0	Bitcoin offline transaction generator

5.2.2 Technology Stack

Table 5.5 provides an overview of the programming language, the technologies and the libraries that are used to implement the proposed framework.

The framework is developed with Java SE 10 (Oracle Java SE Development Kit 10) as programming language, since Java has a huge community and there are many frameworks written for Java. Dependencies and the build lifecycle are managed with Apache Maven. Apache Maven loads required software libraries from a central repository and integrates them into the build of the framework. Spring's RestTemplate is used to consume HTTP services. In order to convert Java objects to JSON and vice versa, Jackson is used. RxJava is a library for composing asynchronous and event-based software components by using observable sequences. The stream topology is developed with RxJava. web3j is a modular, reactive, type safe Java and Android library for communicating with nodes

[1] https://www.oracle.com/technetwork/java/javase/downloads/jdk10-downloads-4416644.html

[2] https://maven.apache.org/

[3] https://spring.io/projects/spring-boot

[4] https://github.com/FasterXML/jackson

[5] https://github.com/ReactiveX/RxJava

[6] https://web3j.io/

[7] https://logback.qos.ch/

[8] https://commons.apache.org/proper/commons-codec/

[9] https://junit.org/junit5/

[10] https://site.mockito.org/

[11] https://github.com/aafomin/gerbera

Table 5.6: The software clients used for running network nodes.

Software	Blockchain
Bitcore[12] 4.1.1	Bitcoin node
Parity[13] 2.0.1	Ethereum and Ethereum Classic node
Gexp[14] 1.7.2	Expanse node

of the Ethereum network via JSON-RPC. Logback is used to format and categorize log output. Apache Commons Code comes into play if hex encoding and decoding is required. Unit tests are written with JUnit Jupiter and mockito. gerba is an offline transaction generator that provides helper methods to generate raw hex Bitcoin transactions.

The software clients that are used for running network nodes (external data sources) are listed in Table 5.6.

These nodes are used by the Metric Collector and by the Data Access Service for retrieving data and for broadcasting new transactions. All nodes expose API endpoints as depicted in Figure 5.2. Those API endpoints that are relevant for this thesis are discussed in Section 5.2.3.

5.2.3 Interface Descriptions

In the following sections, Java interfaces and node API endpoints that are relevant for the implementation of the proposed are described.

IMetricCollector

This interface has to be implemented by each Metric Collector. The following tables provide an overview all interface methods:

Table 5.7: The method getBlockObservable of the Java interface IMetricCollector.

getBlockObservable	
Description	Returns an Observable that emits a stream of block lists, where each list contains all blocks that have been mined during the last 24 hours.
Parameters	None
Return type	Observable<List<Block≫

[12] https://bitcore.io/api/

[13] https://www.parity.io/ethereum/

[14] https://github.com/expanse-org/go-expanse

Table 5.8: The method getAvgBlockTimeObservable of the Java interface IMetricCollector.

getAvgBlockTimeObservable	
Description	Returns an Observable that emits a stream of inter-block times.
Parameters	None
Return type	Observable<Double>

Table 5.9: The method getTransactionThroughputObservable of the Java interface IMetricCollector.

getTransactionThroughputObservable	
Description	Returns an Observable that emits a stream of values that represent the number of transactions processed per second.
Parameters	None
Return type	Observable<Double>

Table 5.10: The method getBlockPercentagePerMinerObservable of the Java interface IMetricCollector.

getBlockPercentagePerMinerObservable	
Description	Returns an Observable that emits a stream of mappings between miner addresses and the corresponding proportions of mined blocks.
Parameters	None
Return type	Observable<Map<String, Double≫

Table 5.11: The method getNetworkHashrateObservable of the Java interface IMetricCollector.

getNetworkHashrateObservable	
Description	Returns an Observable that emits a stream of values representing the network's hash rate.
Parameters	None
Return type	Observable<Double>

Table 5.12: The method getExchangeRateObservable of the Java interface IMetricCollector.

getExchangeRateObservable	
Description	Returns an Observable that emits a stream of values representing the actual price for the underlying cryptocurrency.
Parameters	None
Return type	Observable<BigDecimal>

Table 5.13: The method getCostsForWritingDataObservable of the Java interface IMetricCollector.

getCostsForWritingDataObservable	
Description	Returns an Observable that emits a stream of values representing the costs for writing one KB of data into the blockchain.
Parameters	None
Return type	Observable<BigDecimal>

Table 5.14: The method getCostsForRetrievingDataObservable of the Java interface IMetricCollector.

getCostsForRetrievingDataObservable	
Description	Returns an Observable that emits a stream of values representing the costs for reading one KB of data from the blockchain.
Parameters	None
Return type	Observable<BigDecimal>

Table 5.15: The method getStorageFeeObservable of the Java interface IMetricCollector.

getStorageFeeObservable	
Description	Returns an Observable that emits a stream of values representing the rental fees that have to be paid for one KB of data per hour.
Parameters	None
Return type	Observable<BigDecimal>

IDataAccessService

The IDataAccessService interface provides methods for writing data to and reading data from the underlying blockchain. The following tables describe the methods of the interface:

Table 5.16: The method getData of the Java interface IDataAccessService.

getData	
Description	Returns a list of all data strings that have been written to the underlying blockchain within the given date range.
Parameters	ZonedDateTime from, ZonedDateTime to
Return type	List<String>

Table 5.17: The method writeData of the Java interface IDataAccessService.

writeData	
Description	Writes the given string into the underlying blockchain.
Parameters	String data
Return type	void

Table 5.18: The method writeDataList of the Java interface IDataAccessService.

writeDataList	
Description	Writes all strings of the given list into the underlying blockchain.
Parameters	List<String>
Return type	void

Bitcore

Bitcore supports the Insight API. The Insight API is a Bitcoin HTTP and web socket API service for requesting blockchain information and for submitting transactions. A detailed description of the API endpoints is given at `https://github.com/bitpay/insight-api`. The following tables merely list those API endpoints that are relevant for the implementation of the proposed framework.

Table 5.19: Insight API endpoint for requesting the block hash by its height.

Get Block Hash By Height	
Description	Returns the hash of the block that has the given block number (height).
Request	GET http://node-ip/insight-api/block-index/{height}
Response	JSON object holding the hash of the block.

Table 5.20: Insight API endpoint for requesting a transaction by hash.

Get Transaction By Hash	
Description	Returns the transaction with the given id (hash).
Request	GET http://node-ip/insight-api/tx/{txid}
Response	JSON object representing the requested transaction.

Table 5.21: Insight API endpoint for requesting basic network information.

Get Network Information	
Description	Returns basic information about the Bitcoin network such as protocol version, number of blocks, difficulty, etc.
Request	GET http://node-ip/insight-api/status?q=getInfo
Response	JSON object representing the network information.

Table 5.22: Insight API endpoint for requesting a block by its hash.

Get Block By Hash	
Description	Returns the block with the given hash.
Request	GET http://node-ip/insight-api/block/{hash}
Response	JSON object representing the requested block.

Table 5.23: Insight API endpoint for requesting transactions by address.

Get Transactions By Address	
Description	Returns a list of transactions sent from the given address.
Request	GET http://node-ip/insight-api/txs?address={address}
Response	JSON object representing the list of transactions.

Table 5.24: Insight API endpoint for requesting unspent transaction outputs.

Get Unspent Transaction Outputs By Address	
Description	Returns a list of unspent transaction outputs belonging to the given address.
Request	GET http://node-ip/insight-api/addr/{address}/utxo
Response	JSON object representing the list of unspent transaction outputs.

Table 5.25: Insight API endpoint for sending transactions.

Send Transaction	
Description	Broadcasts the given transaction to the network.
Request	POST http://node-ip/insight-api/tx/send
Request Body	The signed transaction as hex string.
Response	JSON object containing the transaction's hash.

Parity and Gexp

Both Parity and Gexp support common JSON-RPC APIs such as eth, net and web3. A detailed description of these APIs is given at `https://wiki.parity.io/JSONRPC`. The following tables merely list those API endpoints that are relevant for this thesis.

Table 5.26: JSON-RPC endpoint for requesting the transaction count.

eth_getTransactionCount	
Description	Returns the number of transactions sent from an address.
Parameters	An address and a default block parameter.
Response	Integer representing the number of transactions.

Table 5.27: JSON-RPC endpoint for requesting an uncle by block hash and index.

eth_getUncleByBlockHashAndIndex	
Description	Returns the uncle at the given index position of the block identified by the given hash.
Parameters	The hash of a block and the uncle's index position.
Response	Information about the uncle at the given index position of a block.

Table 5.28: JSON-RPC endpoint for requesting a block by its number.

eth_getBlockByNumber	
Description	Returns the block with the given number.
Parameters	Block number and a boolean value indicating whether full transaction objects should be included.
Response	Information about the requested block. In case the boolean parameter is set to true, the block's information structure contains also full transaction objects.

Table 5.29: JSON-RPC endpoint for requesting the current block number.

eth_blockNumber	
Description	Returns the number of the most recent block.
Parameters	None.
Response	The block number as integer.

Table 5.30: JSON-RPC endpoint for requesting the current gas price.

eth_gasPrice	
Description	Returns the current gas price in wei.
Parameters	None.
Response	An integer representing the current gas price.

Table 5.31: JSON-RPC endpoint for sending a transaction.

eth_sendRawTransaction	
Description	Broadcasts a message to the network.
Parameters	The signed transaction data.
Response	The new transaction's hash.

Table 5.32: Parity JSON-RPC endpoint for retrieving pending transactions.

parity_pendingTransactions	
Description	Returns all pending transactions that are known to the node. This API endpoint is only supported by Parity.
Parameters	None.
Response	A list of all pending transactions known to the node.

Table 5.33: Gexp JSON-RPC endpoint for retrieving pending transactions.

eth_pendingTransactions	
Description	Returns all pending transactions sent from one of the addresses that are known to the Gexp node. This API endpoint is only supported by Gexp.
Parameters	None.
Response	A list of pending transactions.

5.3 Implementation

In this section, we will present details about the implementation of the previously designed framework. The next pages cover installation and deployment instructions in order to get the developed framework up and running. These steps include the setup of VMs in the Google Cloud Platform, and the installation and configuration of each network node. The section concludes with Java examples that illustrate how to get the framework up and running.

5.3.1 Deployment and Setup of the Network Nodes

For each supported blockchain, we run a network node that is used by the blockchain's Metric Collector and Data Access Service to interact with the network. These nodes

Table 5.34: Allocated VM resources.

VM (Network Node)	CPU	Memory	Storage
Bitcoin	1 vCPU	4 GB RAM	500 GB HDD
Ethereum	1 vCPU	4 GB RAM	100 GB HDD
Ethereum Classic	1 vCPU	4 GB RAM	50 GB HDD
Expanse	1 vCPU	4 GB RAM	30 GB HDD

are deployed on the Google Cloud Platform. For each node, a separate VM instance with a fresh Ubuntu 18.04.1 LTS installation is created. Table 5.34 shows the resources allocated to each VM instance.

An important aspect to clarify is the reason why we allocated only 100 GB HDD storage to the Ethereum VM and 500 GB HDD storage to the Bitcoin VM. As described in previous sections, we use the Parity software to run an Ethereum network node. Parity offers a flag that is called *warp* and instructs the node to perform the warp synchronization. The warp synchronization allows for an extremely fast synchronization that skips almost all of the block processing and injects downloaded snapshots directly into the database. This mechanism is very fast and needs much less storage than a full synchronization where the entire Ethereum blockchain is downloaded and processed.

It is recommended to allocate more resources in order to accelerate the synchronization phase, e.g., 4 vCPUs and 15 GB RAM. Later on, it is sufficient to allocate those resources that are listed in Table 5.34.

Installation and Configuration of the Bitcoin Node

In order to run a Bitcoin network node, we use the software Bitcore. Listing 5.1 shows the required installation instructions.

Listing 5.1: Installation instructions of Bitcore.

```
1 curl -o-
     https://raw.githubusercontent.com/creationix/nvm/v0.33.11/install.sh
     | bash
2 nvm install v4
3 apt-get install libzmq3-dev build-essential
4 npm install -g bitcore
5 bitcore create mynode
```

In line 1, the Node Version Manager is installed. The Node Version Manager makes it simple to switch between different Node.js versions. The instruction in line 2 installs Node.js v4. In line 3, ZeroMQ is downloaded and installed. Bitcore is installed in line 4. Finally, a new Bitcore node is created in line 5.

In order to run the node, the command *bitcored* is called as shown in Listing 5.2. It is import to change to the *mynode* directory that contains relevant information for running the node.

Listing 5.2: Start instructions of Bitcore.

```
cd mynode
bitcored
```

Listing 5.3 presents the content of the node's configuration file *mynode/bitcore-node.json*.

Listing 5.3: Start instructions of Bitcore.

```
1  {
2    "network": "livenet",
3    "port": 3001,
4    "services": [
5      "bitcoind",
6      "insight-api",
7      "insight-ui",
8      "web"
9    ],
10   "servicesConfig": {
11     "bitcoind": {
12       "spawn": {
13         "datadir": "/home/p_frauenthaler/.bitcore/data",
14         "exec":
     "/home/p_frauenthaler/.nvm/versions/node/v4.9.1/lib/node_modules/
     bitcore/node_modules/bitcore-node/bin/bitcoind"
15       }
16     },
17     "insight-api": {
18       "disableRateLimiter": true
19     }
20   }
21 }
```

If the service *insight-api* is not listed in the configuration file, it must be installed via the command `bitcore install insight-api`. The configuration setting in line 18 disables the rate limiter, otherwise it might be the case that requests sent from the Metric Collector are rejected due to a violation of a rate limit.

Installation and Configuration of the Ethereum Node and Ethereum Classic Node

For running an Ethereum and an Ethereum Classic node, we use Parity. On Ubuntu, Parity can be installed with a single command: `bash <(curl https://get.parity.io -L)`

After the successful installation the node can be started without further configuration. A node can be started with the following command:

Table 5.35: Relevant flags for starting an Ethereum or Ethereum Classic node.

Flag	Meaning
chain	Specifies the blockchain to use. *mainnet* is the main Ethereum network and *classic* specifies the Ethereum Classic network.
warp	Instructs the node to perform the warp synchronization. The warp synchronization allows for an extremely fast synchronization that skips almost all of the block processing and injects downloaded snapshots directly into the database.
cache-size	Sets the total amount of discretionary memory (in MB) to use.
jsonrpc-apis	Specifies the APIs available through the HTTP JSON-RPC interface.
jsonrpc-interface	Specifies the hostname portion of the HTTP JSON-RPC API. This flag accepts an interface's IP address, *all* (all interfaces) or *local*. Default is local.

```
parity --chain=mainnet --warp --cache-size 1024 --jsonrpc-apis "eth,parity" --jsonrpc-interface
all
```

In order to run an Ethereum Classic node, the argument parameter *classic* must be used instead of *mainnet*.

In Table 5.35, the relevant flags are listed and their meaning is explained.

Installation and Configuration of the Expanse Node

In order to setup and run an Expanse node, Gexp, an official golang implementation of the Expanse protocol, is used. Building Gexp requires both a Go installation (version 1.7 or later) and a C compiler. Gexp can be built with `make gexp`. After the compilation process has been completed successfully, the binary for running Gexp is located in the directory *build/bin/*. An Expanse node is started with the following command:

```
gexp --cache=2048 --rpc --rpcport 9656 --rpcaddr "0.0.0.0" --rpcapi "eth,net,web3"
```

Explanations of the used flags are given in Table 5.36.

Instructions for Running the Framework

As described in Section 5.2.2, the framework is developed with Java. Thus, Java must be installed on the system in order to get the framework running. Listing 5.4 shows how an instance of the Blockchain Manager is configured and created.

Table 5.36: Relevant flags for starting an Expanse node.

Flag	Meaning
cache	Bumps the memory allowance of the database to the specified size (in MB).
rpc	Enables the HTTP-RPC server.
rpcport	Sets the port of the HTTP-RPC server.
rpcapi	Specifies the APIs available through the HTTP-RPC interface (default: eth, net, web3).
rpcaddr	Specifies HTTP-RPC server listening interface (default: "localhost").

Listing 5.4: Instructions for building an instance of the Blockchain Manager.

```
1  blockchainManager = BlockchainManager
2    .newInstance()
3    .setRankingSettings(rankingSettings)
4    .setThresholdValidationSettings(validationSettings)
5    .preSelectBlockchain(currentBlockchain)
6    .addBlockchain(bitcoinBlockchain)
7    .addBlockchain(ethereumBlockchain)
8    .addBlockchain(ethereumClassicBlockchain)
9    .addBlockchain(expanseBlockchain)
10   .build();
```

In lines 3 - 4, the settings that affect the weighted ranking system and the threshold validation mechanism are set. A blockchain is registered to the Blockchain Manager by calling the method addBlockchain(). The method build() initializes and starts the Blockchain Manager. Listing 5.5 outlines how to create a blockchain.

Listing 5.5: Instructions for creating a blockchain.

```
IMetricCollector bitcoinMetricCollector = ...;
IDataAccessService bitcoinDataAccessService = ...;
BlockchainMetaData bitcoinBlockchain = new BlockchainMetaData();
bitcoin.setIdentifier("Bitcoin");
bitcoin.setMetricCollector(bitcoinMetricCollector);
bitcoin.setDataAccessService(bitcoinDataAccessService);
bitcoin.setNumberOfRequiredConfirmations(6);
bitcoin.setReputation(8);
```

The methods setNumberOfRequiredConfirmations() and setReputation() are used to specify metrics M9 and M10, since these metrics can not be computed automatically.

In order to retrieve switchover suggestions from the Blockchain Manager, a subscription to the corresponding Observable as shown in Listing 5.6 is necessary.

Listing 5.6: Instructions for receiving switchover suggestions.

```
blockchainManager
  .getSwitchoverSuggestionObservable()
  .subscribe(switchoverSuggestion -> {
    // logic that should be executed for each switchover suggestion
  },
  throwable -> LOG.error(throwable.getMessage(), throwable)
);
```

In this chapter, functional requirements, the technical design and implementation details of the proposed framework have been introduced. In the next chapter, the reference implementation of the framework is analyzed and evaluated in the context of different scenarios.

CHAPTER 6

Framework Evaluation

This chapter presents the evaluation of the designed and developed framework based on an evaluation setup and defined evaluation scenarios. Evaluation is the process to assess whether an outcome serves the aimed purpose and fulfills the specified requirements. The chapter starts off with a specification of the evaluation setup in Section 6.1, followed by evaluation results for the Bitcoin, Ethereum, Ethereum Classic and Expanse blockchains. In Section 6.3, different evaluation scenarios for the analysis of the developed framework are presented.

6.1 Evaluation Setup

The evaluation setup defines the infrastructure that is used to evaluate the developed framework. As outlined in Section 5.2, the proposed framework relies on external nodes that are used to communicate with the corresponding networks (e.g., the Bitcoin network). For each network node, a separate VM is deployed on the Google Cloud Platform. Table 6.1 lists the allocated VM resources. The nodes are started with the instructions described in Section 5.3.

Table 6.1: Evaluation Setup: Allocated resources for VMs running on the Google Cloud Platform.

VM (Network Node)	CPU	Memory	Storage
Bitcoin	1 vCPU	4 GB RAM	500 GB HDD
Ethereum	1 vCPU	4 GB RAM	100 GB HDD
Ethereum Classic	1 vCPU	4 GB RAM	50 GB HDD
Expanse	1 vCPU	4 GB RAM	30 GB HDD

The framework itself is executed on a MacBook Pro (Retina, 13-inch, Late 2013) with the following hardware and software equipment:

- Processor: 2.4 GHz Intel Core i5
- Memory: 8 GB 1600 MHz DDR3
- Graphic Card: Intel Iris 1536 MB
- SSD: 256 GB
- OS: macOS High Sierra (version 10.13.6)
- Java Virtual Machine: Oracle Java SE Development Kit 10

The Blockchain Manager is initialized with the Bitcoin, Ethereum, Ethereum Classic and Expanse blockchains. Furthermore, the Blockchain Manager can be configured with arbitrary settings for the weighted ranking system (see Section 5.1), since the weighted ranking is only relevant for the Scenarios 5 and 11 (see Sections 6.3.5 and 6.3.11). Concrete threshold validation settings are described in the Sections 6.3.7 and 6.3.8.

6.2 Measured Blockchain Metrics

In this section, we present metric values for Bitcoin, Ethereum, Ethereum Classic and Expanse, since the developed framework supports these blockchains by default. The values serve as basis for the evaluation scenario described in Section 6.3.11 and have been measured on every second day between 25.09.2018 and 17.10.2018 by using the developed framework. Since read operations are free and storage fees are not implemented in Bitcoin, Ethereum, Ethereum Classic and Expanse at the time of writing this thesis, the corresponding metrics are not listed in this section. Furthermore, a blockchain's reputation and the number of required block confirmations are not presented, because these metric values have to be provided by the framework user.

Figure 6.1 illustrates the progression of the costs for writing one kilobyte of data into a blockchain. As shown in the figure, storing data in Bitcoin is the most expensive option, whereas the costs for storing data in Ethereum Classic or Expanse are very close to zero. Table 6.2 presents for each blockchain the average value and the standard deviation.

Figure 6.1: A depiction of the costs for writing one kilobyte of data into the Bitcoin, Ethereum, Ethereum Classic and Expanse blockchain (measured between 25.09.2018 and 17.10.2018).

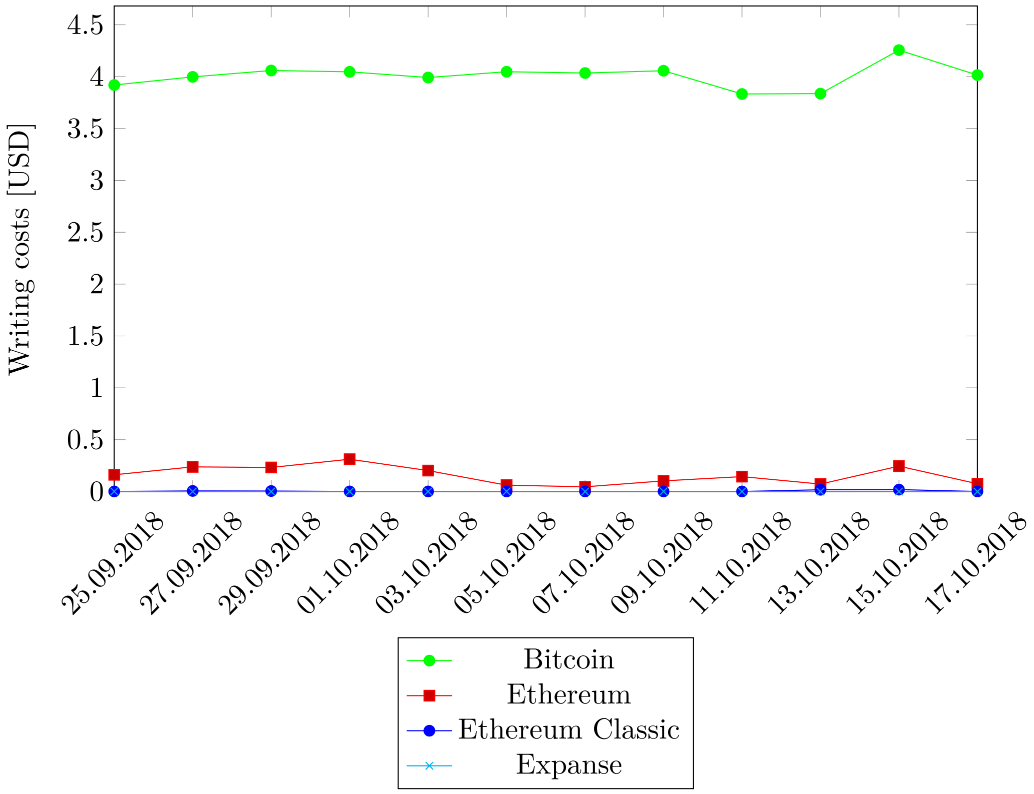


Table 6.2: The average value and standard deviation of the costs for writing one kilobyte of data into the Bitcoin, Ethereum, Ethereum Classic and Expanse blockchain.

Blockchain	Average Value [USD]	Standard Deviation [USD]
Bitcoin	4.00798	0.10743
Ethereum	0.15776	0.08387
Ethereum Classic	0.00460	0.00620
Expanse	0.00038	0.00063

In Figure 6.2, the progression of each blockchain's inter-block time is shown. The values are shown on a logarithmic scale. With an average time of 10.15 minutes, Bitcoin has the greatest block time. Ethereum and Ethereum Classic have almost the same average block time of about 14 seconds. Expanse has an average inter-block time of 44 seconds. The average values and the standard deviations are outlined in Table 6.3.

Figure 6.2: A depiction of the inter-block times for Bitcoin, Ethereum, Ethereum Classic and Expanse (measured between 25.09.2018 and 17.10.2018).

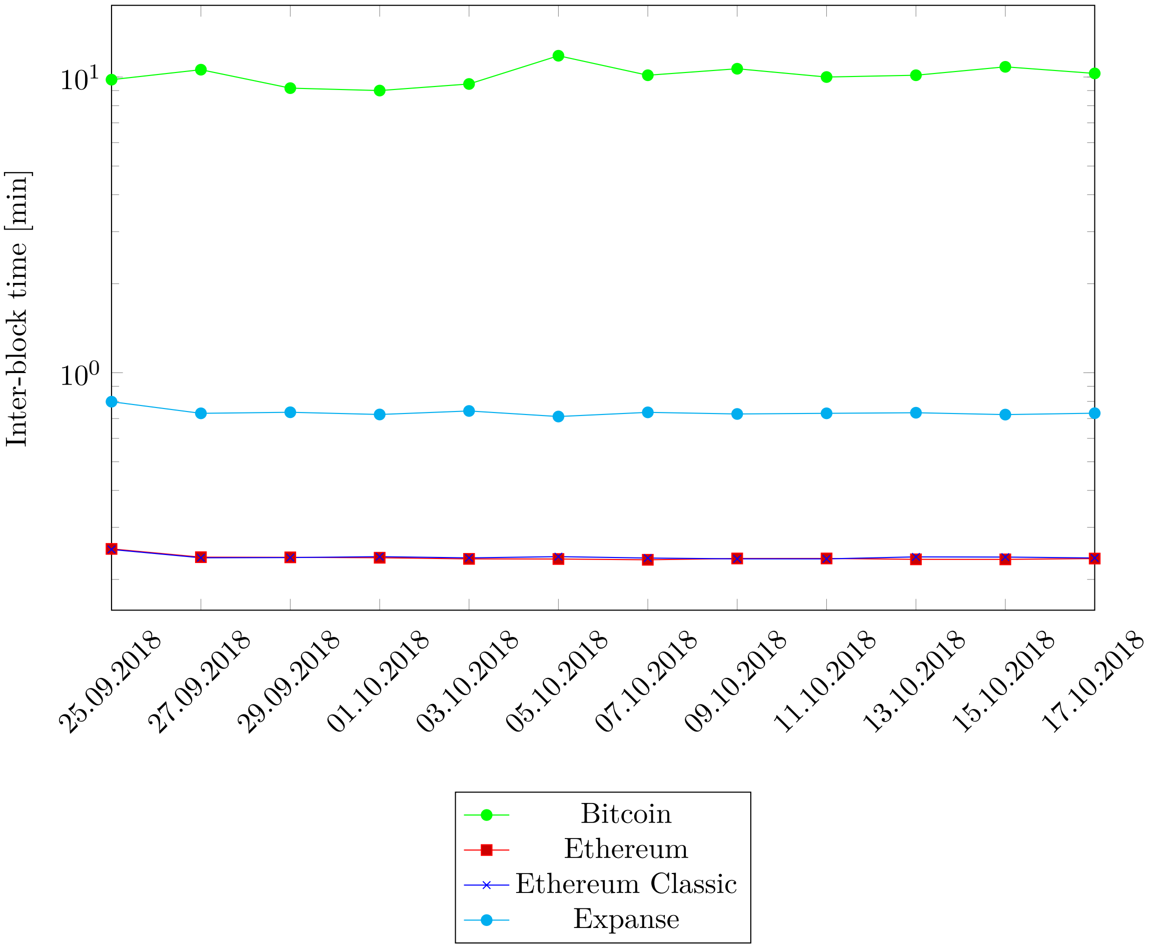


Table 6.3: The average value and standard deviation of each blockchain's inter-block times.

Blockchain	Average Value [s]	Standard Deviation [s]
Bitcoin	609.47	44.34
Ethereum	14.21	0.32
Ethereum Classic	14.30	0.27
Expanse	44.06	1.24

Figure 6.3 shows the progression of the exchange rates between USD and each blockchain's underlying cryptocurrency. Again, values are shown on a logarithmic scale. As shown in

the figure, Bitcoin is the most expensive cryptocurrency, whereas Expanse is the cheapest one. Table 6.4 presents for each blockchain the average value and the standard deviation.

Figure 6.3: A depiction of the exchanges rates for Bitcoin, Ethereum, Ethereum Classic and Expanse (measured between 25.09.2018 and 17.10.2018).

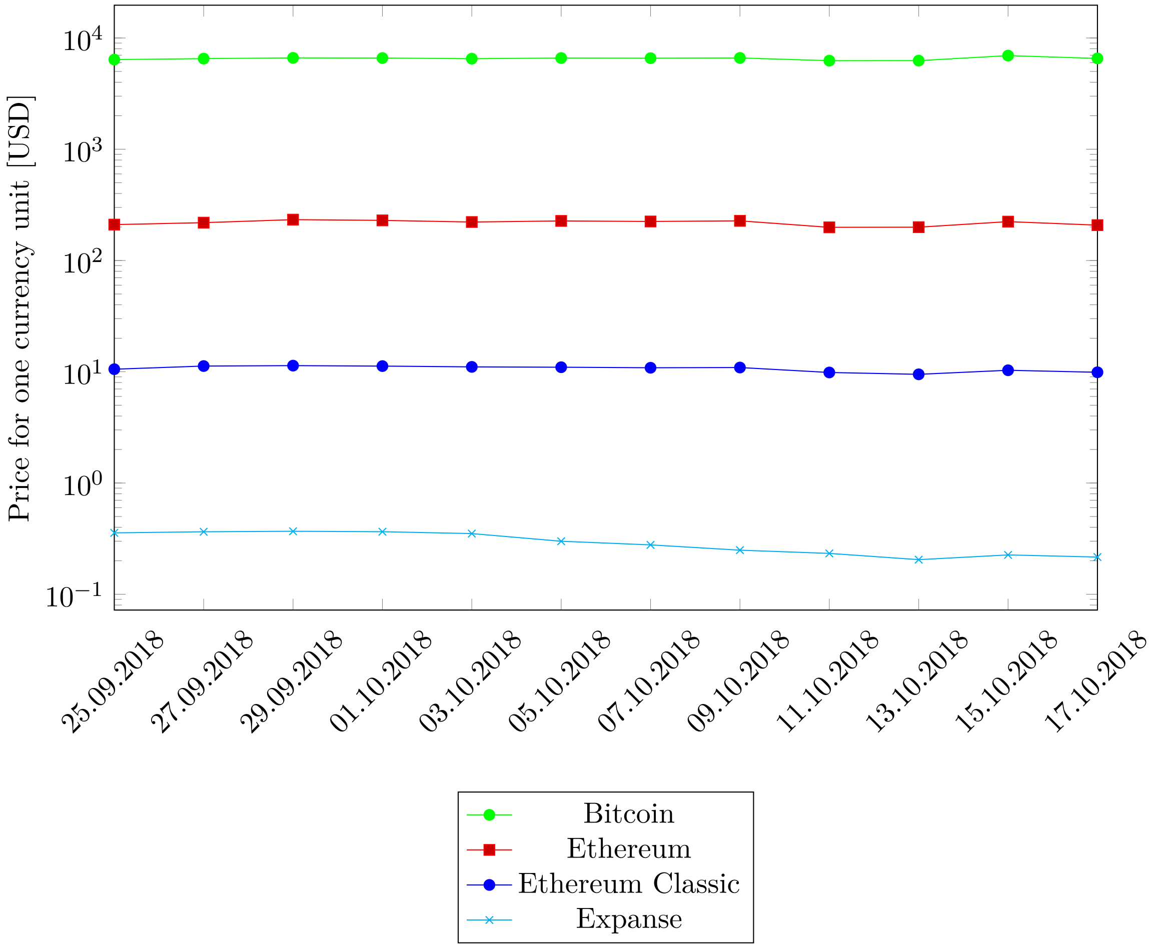


Table 6.4: The average value and standard deviation of the exchange rates between USD and each blockchain's underlying cryptocurrency.

Blockchain	Average Value [USD]	Standard Deviation [USD]
Bitcoin	6,538.44	175.23
Ethereum	218.35	11.13
Ethereum Classic	10.64	0.60
Expanse	0.29	0.06

The progression of each blockchain's transaction throughput is presented in Figure 6.4. As shown in the depiction, Ethereum has the highest transaction throughput, whereas Expanse processes the lowest number of transactions per second. Each blockchain's average value and standard deviation are outlined in Table 6.5.

Figure 6.4: A depiction of the transaction throughputs for Bitcoin, Ethereum, Ethereum Classic and Expanse (measured between 25.09.2018 and 17.10.2018).

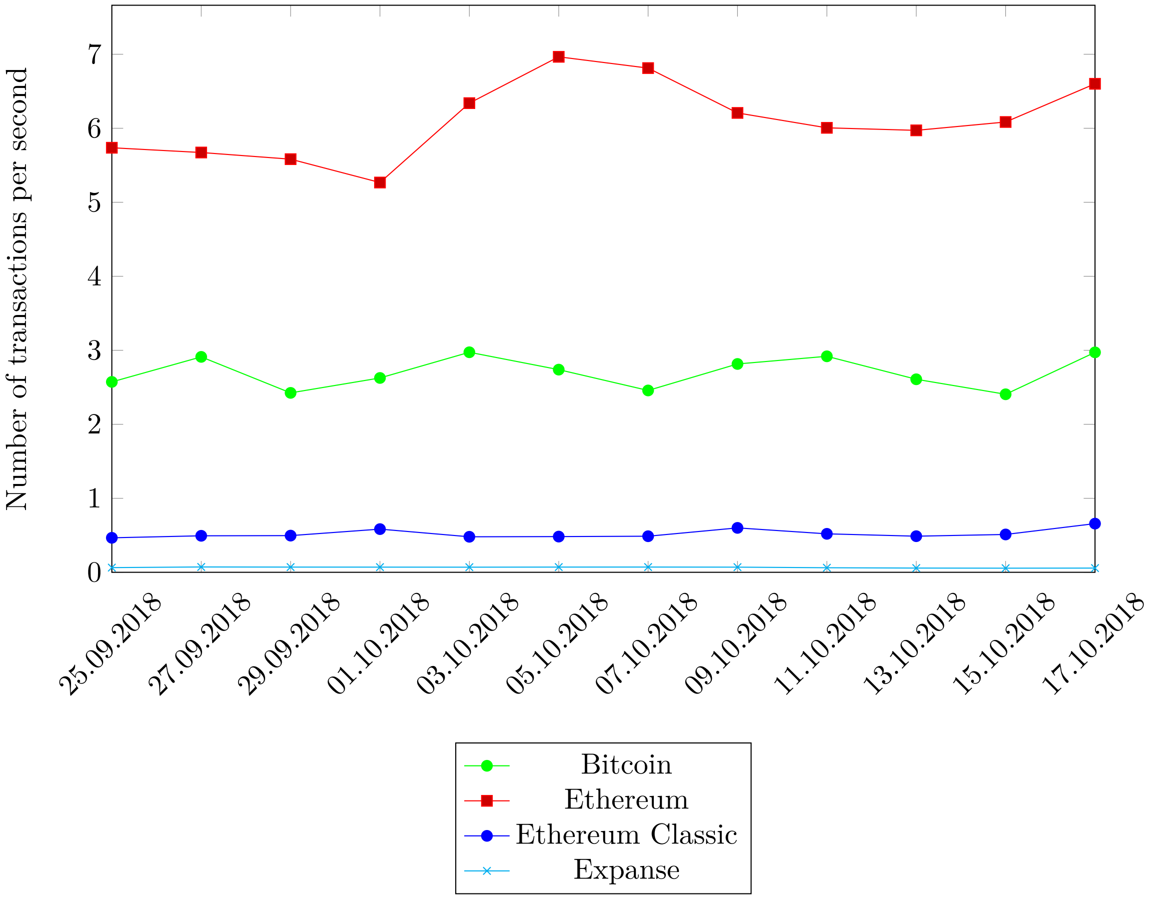


Table 6.5: The average value and standard deviation of each blockchain's transaction (abbr. tx) throughputs.

Blockchain	Average Value [tx/s]	Standard Deviation [tx/s]
Bitcoin	2.70	0.21
Ethereum	6.10	0.49
Ethereum Classic	0.52	0.06
Expanse	0.07	0.01

Figure 6.5 presents the progression of each blockchain's hash rate. The numbers are given on a logarithmic scale. As outlined in the depiction, the Bitcoin network calculates by far the most hashes per second, whereas the Expanse network has the lowest hash rate. In Table 6.6, each blockchain's average value and standard deviation are presented.

Figure 6.5: A depiction of the network hash rates of Bitcoin, Ethereum, Ethereum Classic and Expanse (measured between 25.09.2018 and 17.10.2018).

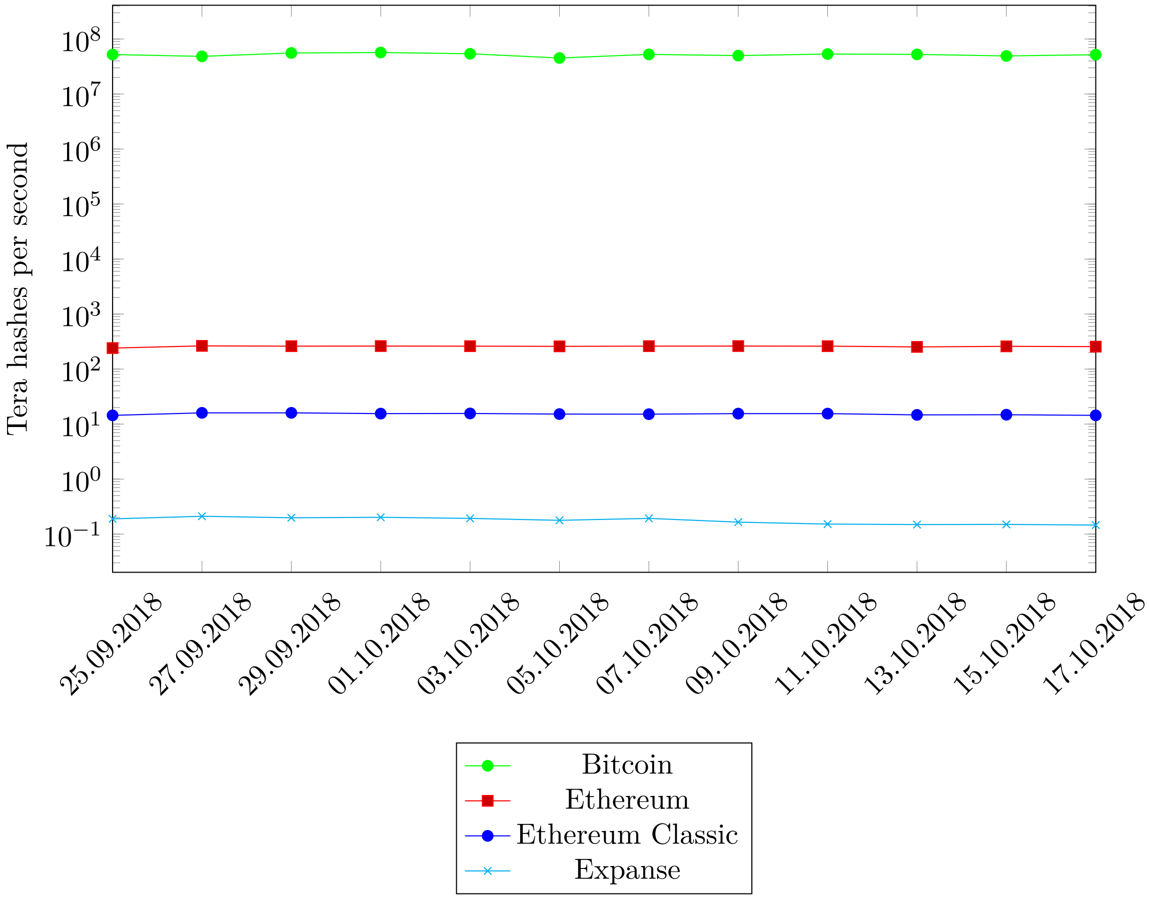
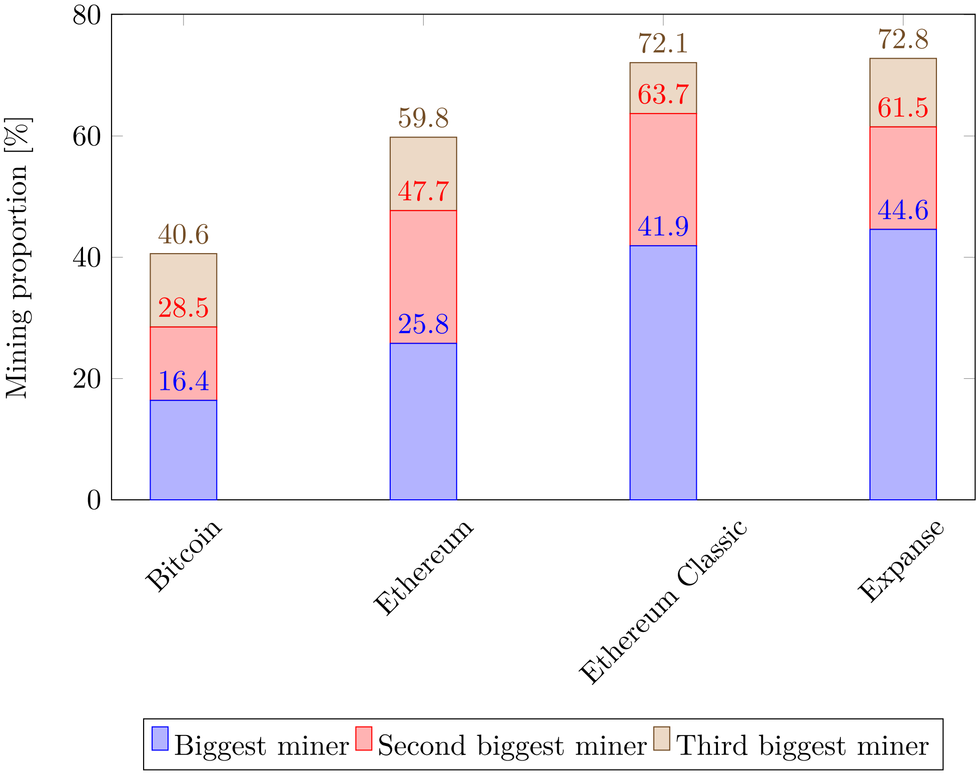


Table 6.6: The average value and standard deviation of each blockchain's hash rate in tera hashes per second (abbr. TH/s).

Blockchain	Average Value [TH/s]	Standard Deviation [TH/s]
Bitcoin	51,868,630.60	3,102,978.96
Ethereum	258.36	5.95
Ethereum Classic	15.29	0.53
Expanse	0.07	0.01

Figure 6.6 shows for each blockchain the distribution of the network's hash power. For a better overview, only the proportions of the three biggest miners of each network are presented. A remarkable aspect is that the biggest miner of the Ethereum Classic network controls more than 40 % of the overall hash power and the biggest Expanse miner computes approximately 44.6 % of all hashes.

Figure 6.6: A depiction of the distribution of the hash power among the three biggest miner (measured on 17.10.2018).



6.3 Evaluation Scenarios and Results

In this section, eleven evaluation scenarios that serve as basis for a holistic evaluation of the developed framework are presented. These experimental evaluation scenarios range from the evaluation of features and mechanisms to more specific assessments of resource consumption and switchover times. Furthermore, the results for the presented scenarios are described and illustrated.

6.3.1 Scenario 1: Blockchain Manager – Concurrent Write Operations

Description

As described in Section 5.2, the Blockchain Manager exposes a method for writing data into the currently selected blockchain. The experiment of this evaluation scenario analyzes the case when multiple threads simultaneously deliver data to the exposed method. All threads hold a reference to the same Blockchain Manager instance. The configuration of the Blockchain Manager that is described in Section 6.1 remains unchanged for this experiment. Handling concurrent write operations is the Blockchain Manager's responsibility. Therefore, the experiment is independent from the preselected blockchain (i.e., the blockchain that is used by the Blockchain Manager for write operations). Since the gas price and thus write operations are very cheap, Expanse is used as the preselected chain for the Blockchain Manager. In order to assess the Blockchain Manager's reaction to concurrent write operations, a thread pool of ten threads is created. Each thread delivers its name to the Blockchain Manager through the exposed method. The expected result is that the Blockchain Manager ensures that no data string gets lost, is corrupted or mixed with characters of other data strings. It is expected that always only one thread is allowed to write data into the preselected chain of the Blockchain Manager at the same time. One conduction of this experiment is sufficient, since the deterministic functionality of the framework is analyzed and the result is independent from gathered blockchain metrics.

Result

Listing 6.1 shows an extraction of the framework's log output. As shown in the listing, only one thread is allowed to write data at the same time. There is no interleaving in the log output indicating that a thread enters the write method before another thread has exited it. Furthermore, for each input string (i.e., thread name), an Expanse transaction is created, signed and sent to the network. The transaction data can be verified at `https://gander.tech/`. Each transaction contains the correct thread name, no characters got lost, corrupted or were mixed with others.

Listing 6.1: Log extraction that illustrates the handling of concurrent write operations.

```
10:43:05,189 [pool-6-thread-1] - Entered write data method
10:43:05,286 [pool-6-thread-1] - Store data: pool-6-thread-1
10:43:05,373 [pool-6-thread-1] - Successfully sent transaction:
     0x5ce800b5c083b1a3f304be6befc91cc3641e5acddefbb14bb0815bcd34829fae

10:43:05,374 [pool-6-thread-1] - Exit write data method

10:43:05,374 [pool-6-thread-10] - Entered write data method
10:43:05,432 [pool-6-thread-10] - Store data: pool-6-thread-10
10:43:05,500 [pool-6-thread-10] - Successfully sent transaction:
     0xe8958e9e0675aa3436e0000a46e5de19a710c093f655f3fb9726ff5d5d297b40
10:43:05,500 [pool-6-thread-10] - Exit write data method

10:43:05,501 [pool-6-thread-9] - Entered write data method
10:43:05,557 [pool-6-thread-9] - Store data: pool-6-thread-9
10:43:05,621 [pool-6-thread-9] - Successfully sent transaction:
     0xce790d4f6fd41c7b1bcaf41a953c2b2c5b1f6f3d2407ff188cbd53b37699c172
10:43:05,621 [pool-6-thread-9] - Exit write data method

10:43:05,621 [pool-6-thread-8] - Entered write data method
10:43:05,680 [pool-6-thread-8] - Store data: pool-6-thread-8
10:43:05,772 [pool-6-thread-8] - Successfully sent transaction:
     0xbf0dd8ff9daf73fc902c844fc5426987306874aaf1710d9a8028612bb46e6ccb
10:43:05,772 [pool-6-thread-8] - Exit write data method

10:43:05,772 [pool-6-thread-7] - Entered write data method
10:43:05,838 [pool-6-thread-7] - Store data: pool-6-thread-7
10:43:05,906 [pool-6-thread-7] - Successfully sent transaction:
     0xe21f1033de48576255f037159aa09ac0ce53efbfbb0fab64324f6483a9bd288c
10:43:05,906 [pool-6-thread-7] - Exit write data method
```

6.3.2 Scenario 2: Blockchain Manager – Write Operations with Large Amounts of Data

Description

As stated in Section 5.2.1, data is stored in transactions rather than smart contracts due to the lack of smart contracts in Bitcoin and because this is the cheaper option for Ethereum-based blockchains. This scenario shows the developed mechanisms of how the framework reacts when the input data string is too long in order to get stored in a single transaction. For this purpose, each blockchain's Data Access Service accepts a user-defined string formatter function that decides how the input string should be split up based on the maximum data length. The string formatter function used in this experiment is shown in Listing 6.2. The function splits up the input string into chunks of the maximum allowed length. Expanse is selected as the currently used blockchain for the same reasons stated in the previous scenario. The maximum length is defined with three bytes. The input string used for this experiment is *MasterThesis2018*.

Listing 6.2: String formatter function for Scenario 2.

```
(string, maxLength) -> {
  final int numberOfChunks = (int) Math.ceil(((double)
    string.length()) / maxLength);

  return IntStream
    .range(0, numberOfChunks)
    .mapToObj(index -> string.substring
      (
        index * maxLength
        Math.min((index + 1) * maxLength, string.length())
      )
    )
    .collect(Collectors.toList());
}
```

The expected result is that the input string is split up into the following chunks: *Mas*, *ter*, *The*, *sis*, *201*, *8*. It is expected that for each chunk an Expanse transaction is submitted. One conduction of this experiment is sufficient, since the deterministic functionality of the framework is analyzed and the result is independent from gathered blockchain metrics.

Result

Listing 6.3 shows that the the input string is split up into the expected chunks. For each chunk, a separate transaction is created.

Listing 6.3: Log extraction that illustrates the handling of too large input strings.

```
11:31:53,230 [main] – Store data: Mas
11:31:53,335 [main] – Successfully sent transaction:
    0xbbaeb87a73d2177e5b9c60a4b96825c100e78ae685ec0352fa74738d424d99ff
11:31:53,364 [main] – Store data: ter
11:31:53,431 [main] – Successfully sent transaction:
    0xf005a3f204bff2af495435a19979cd0ef3e74afa446e4cc2910703fa974eed11
11:31:53,459 [main] – Store data: The
11:31:53,523 [main] – Successfully sent transaction:
    0x66fe67d28f113e69c254726f9bf136312e08e34921c489b9c77a0d7267506df9
11:31:53,551 [main] – Store data: sis
11:31:53,616 [main] – Successfully sent transaction:
    0x0d64c89651b736addbffbaa833cb417249d6680e010809f182a7bd480b506825
11:31:53,646 [main] – Store data: 201
11:31:53,709 [main] – Successfully sent transaction:
    0x80200ff6272974facb10cc06b008be081be869efd0d980f846b33535457d32cd
11:31:53,741 [main] – Store data: 8
11:31:53,816 [main] – Successfully sent transaction:
    0xb52ffa996fc2aa990c7f0873efab105c0882b4682d5084fd8b6e28eee1299396
```

6.3.3 Scenario 3: Write Operations during Switchover

Description

It might take some time until all data is moved to the destination blockchain during the execution of a switchover. The time for a switchover strongly depends on the underlying blockchain and the specified time span that defines the amount of data to transfer. This scenario investigates the case when data is written to the currently used blockchain during the execution of a switchover. In order to conduct this experiment, the Ethereum blockchain is selected as the currently used blockchain (which is used by the Blockchain Manager as destination chain for write operations and source chain for a switchover), since the Ethereum blockchain contains a lot of data and the Ethereum node is not capable of returning all transactions sent from a particular address. Thus, retrieving data from Ethereum takes more time than for Bitcoin, Ethereum Classic and Expanse. With a time span of one hour it is ensured that there is enough time for threads to execute write operations until the switchover is finished. Furthermore, a thread pool of five threads is created. Each of these threads writes its name into the currently used blockchain by calling a method exposed by the Blockchain Manager. The Expanse blockchain is used as destination blockchain due to its low costs for write operations. The expected result is that no data string is written to the currently used blockchain (i.e., Ethereum). All data inputs must be buffered and after the execution of the switchover has been finished, the buffered input strings are written to the new blockchain (i.e., Expanse). One conduction of this experiment is sufficient, since the deterministic functionality of the framework is analyzed and the result is independent from gathered blockchain metrics.

Result

Listing 6.4 shows the buffering of write operations which are performed during the execution of a switchover. After the switchover has been finished, the buffer is flushed which results in five new Expanse transactions (lines 18 - 27). No data is written to the Ethereum blockchain.

Listing 6.4: Log extraction that illustrates the handling of write operations during the execution of a switchover.

```
 1 18:34:48,932 [Thread-36] - start switchover
 2 18:34:53,925 [pool-6-thread-1] - Entered write data method
 3 18:34:53,934 [pool-6-thread-1] - write pool-6-thread-1 to buffer
 4 18:34:53,934 [pool-6-thread-1] - Exit write data method
 5 18:34:53,934 [pool-6-thread-5] - Entered write data method
 6 18:34:53,935 [pool-6-thread-5] - write pool-6-thread-5 to buffer
 7 18:34:53,935 [pool-6-thread-5] - Exit write data method
 8 18:34:53,935 [pool-6-thread-4] - Entered write data method
 9 18:34:53,936 [pool-6-thread-4] - write pool-6-thread-4 to buffer
10 18:34:53,936 [pool-6-thread-4] - Exit write data method
11 18:34:53,936 [pool-6-thread-3] - Entered write data method
12 18:34:53,937 [pool-6-thread-3] - write pool-6-thread-3 to buffer
13 18:34:53,937 [pool-6-thread-3] - Exit write data method
```

```
18:34:53,937 [pool-6-thread-2] - Entered write data method
18:34:53,937 [pool-6-thread-2] - write pool-6-thread-2 to buffer
18:34:53,937 [pool-6-thread-2] - Exit write data method
18:35:29,844 [Thread-36] - switchover finished
18:35:29,906 [Thread-36] - Store data: pool-6-thread-1
18:35:29,996 [Thread-36] - Successfully sent transaction:
    0x9c6ae7f5d3e9a0c5bf1bdb74e5a14c636fe7ef9170cb951e26398b02913b3eb7
18:35:30,053 [Thread-36] - Store data: pool-6-thread-5
18:35:30,121 [Thread-36] - Successfully sent transaction:
    0x1b820cc2ffeab9cd7551f8ca41cd2b546e1b6ab67afbb372625cc4a535d44c30
18:35:30,180 [Thread-36] - Store data: pool-6-thread-4
18:35:30,249 [Thread-36] - Successfully sent transaction:
    0x4eb2c5bb320c720f00b2b8f54dd762caf0ec4af4749df2fbfd1da1586ce29c9c
18:35:30,307 [Thread-36] - Store data: pool-6-thread-3
18:35:30,374 [Thread-36] - Successfully sent transaction:
    0xd014689c19856935494b1849d97c94ed679dfa88f2d804707d2b20069cee5ce4
18:35:30,431 [Thread-36] - Store data: pool-6-thread-2
18:35:30,501 [Thread-36] - Successfully sent transaction:
    0x81e54d58fa71bce523e8d8c6880a34282012d12af0b2aa7446c2301a7a221969
```

6.3.4 Scenario 4: New Switchover during the Execution of a Switchover

Description

The fourth experimental scenario evaluates the situation when a new switchover is started during the execution of a current switchover. For this experiment, the settings of Scenario 3 are used due to the reasons outlined in the previous scenario. The second switchover is started from a different thread. The expected result is that the Blockchain Manager starts the execution of the second switchover only after the execution of the first switchover is finished. There can be only one switchover execution at the same time. One conduction of this experiment is sufficient, since the deterministic functionality of the framework is analyzed and the result is independent from gathered blockchain metrics.

Result

Listing 6.5 shows that the second switchover is blocked until the first one is finished. Only one switchover is executed at the same time.

Listing 6.5: Log extraction that illustrates the handling of concurrent switchovers.

```
18:53:01,356 [Thread-36] - first switchover call
18:53:01,370 [Thread-36] - start switchover
18:53:06,357 [Thread-37] - second switchover call
18:55:23,611 [Thread-36] - switchover finished
18:55:23,612 [Thread-37] - start switchover
18:55:24,112 [Thread-37] - switchover finished
```

6.3.5 Scenario 5: Frequent Changes in the Blockchain Ranking

Description

The experiment of this evaluation scenario analyzes the case when at least two blockchains are very close together in the weighted ranking system, i.e., they have almost the same weighted score. It is investigated how the framework reacts on frequent changes in the blockchain ranking, i.e., frequent changes in the decision which blockchain is the most beneficial one. In order to conduct this experiment, the weighted ranking algorithm is manipulated such that Ethereum Classic and Expanse are alternating with each other in the weighted ranking system, i.e., at one time Ethereum Classic is the most beneficial chain and after at least five seconds Expanse is the most beneficial chain, and so forth. The Blockchain Manager is configured with a weighted ranking timespan of 30 seconds. Thus, it is expected that a switchover is suggested only every 30 seconds. One conduction of this experiment is sufficient, since the deterministic functionality of the framework is analyzed and the result is independent from gathered blockchain metrics.

Result

Listing 6.6 illustrates the handling of frequent changes in the weighted ranking. A switchover suggestion is only received every 30 seconds, regardless of how many changes in the weighted ranking system have been recognized since the last switchover suggestion.

Listing 6.6: Log extraction that illustrates the handling of frequent ranking changes.

```
10:38:55,537 – Most beneficial blockchain: Expanse
10:39:00,677 – Switchover suggestion received: Expanse
10:39:05,367 – Most beneficial blockchain: Ethereum Classic
10:39:10,665 – Most beneficial blockchain: Expanse
10:39:23,642 – Most beneficial blockchain: Ethereum Classic
10:39:30,013 – Most beneficial blockchain: Expanse
10:39:30,679 – Switchover suggestion received: Expanse
10:39:35,557 – Most beneficial blockchain: Ethereum Classic
10:39:40,559 – Most beneficial blockchain: Expanse
10:39:46,304 – Most beneficial blockchain: Ethereum Classic
10:39:53,820 – Most beneficial blockchain: Expanse
10:40:00,031 – Most beneficial blockchain: Ethereum Classic
10:40:00,682 – Switchover suggestion received: Ethereum Classic
```

6.3.6 Scenario 6: Resource Consumption

Description

This scenario investigates the resource consumption of the designed and implemented framework. External data sources are not analyzed, since they are not part of the proposed framework. For conducting this experiment, a fresh instance of the Blockchain Manager is created and started based on the evaluation setup. No further instances of the Blockchain Manager are running. In order to get the entire internal stream topology invoked, a subscription for receiving switchover suggestions is implemented as shown in Listing 6.7. This experiment analyzes the CPU utilization, the Heap size and the number of used threads. These metrics are gathered with VisualVM[1]. After the framework has been started, the metrics are gathered for the next ten minutes (600 seconds). The measurement is conducted five times.

Listing 6.7: Subscription to receive switchover suggestions.

```
blockchainManager
  .getSwitchoverSuggestionObservable()
  .subscribe(suggestion -> {
      String identifier = suggestion.getNextBlockchainResult()
        .getBlockchain().getIdentifier();
      LOG.info("Suggested blockchain: " + identifier);
    },
    throwable -> LOG.error(throwable.getMessage(), throwable)
  );
```

Result

Figure 6.7 shows the progression of the framework's average CPU utilization and Figure 6.8 presents the corresponding standard deviations. In the first minute, the framework's CPU utilization reaches about 52 %. Later on, the CPU utilization decreases and with a few exceptions, remains under five percentage. The rapid increase during the first minute occurs due to the initialization phase. Immediately after the framework has been started, for each blockchain the blocks that have been mined during the last 24 hours are downloaded and processed. For instance, there are approximately 6,000 blocks that have to be downloaded and processed alone for Ethereum. After about 90 seconds, this initialization phase is finished which results in a lower CPU utilization.

In Figure 6.9, the average allocated heap space along with the progression of the average used space is presented. The corresponding standard deviations are shown in Figure 6.10.

Figure 6.11 highlights the average number of live threads and average the number of daemon threads created during the executions of the implemented framework. The corresponding standard deviations are presented in Figure 6.12. The rapid increase of live threads during the first 90 seconds occurs due to the same reasons outlined above for the CPU utilization. The green line shows that threads which are no longer needed for the execution of subsequent tasks are killed in order to decrease the framework's resource consumption.

[1] https://visualvm.github.io/

Figure 6.7: The graph shows the progression of the framework's average CPU utilization (measured on 28.10.2018).

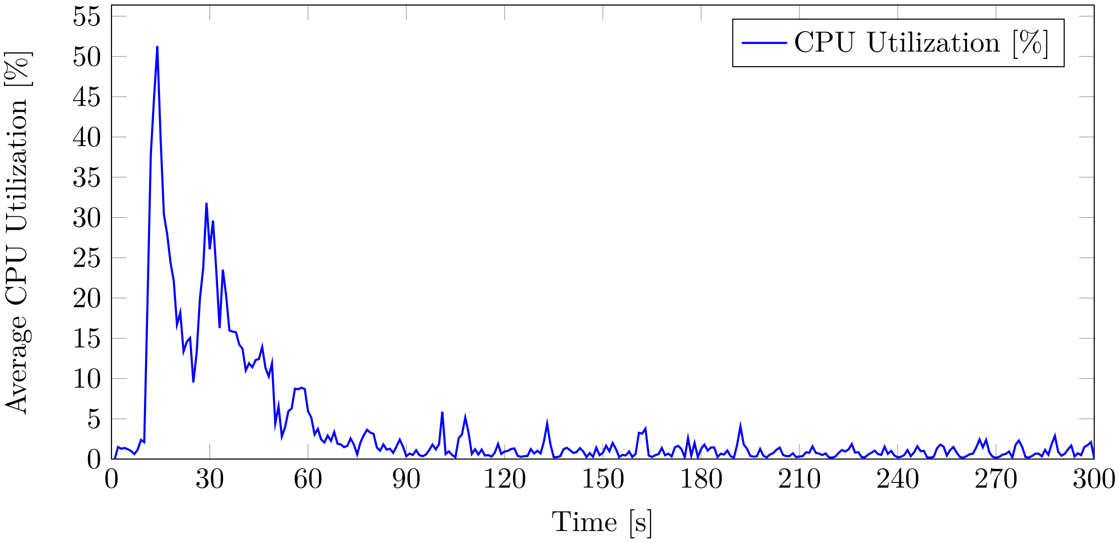


Figure 6.8: The graph shows the standard deviations of the CPU utilization (measured on 28.10.2018).

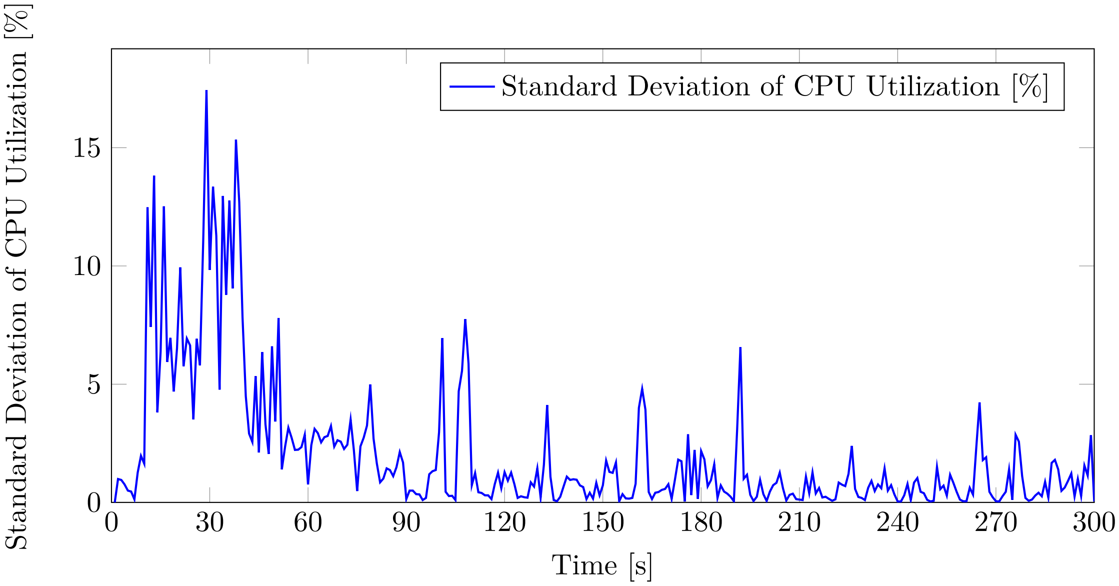


Figure 6.9: The graphs show the progression of the average allocated heap size and of the average heap usage.

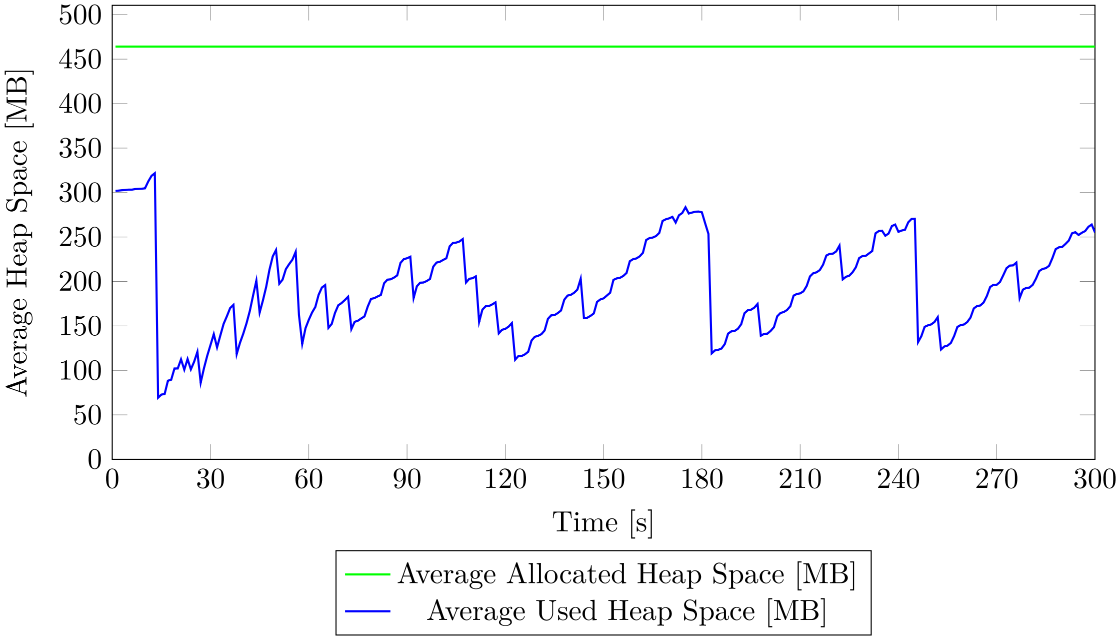


Figure 6.10: The graphs show the standard deviations of the allocated heap sizes and of the heap usage.

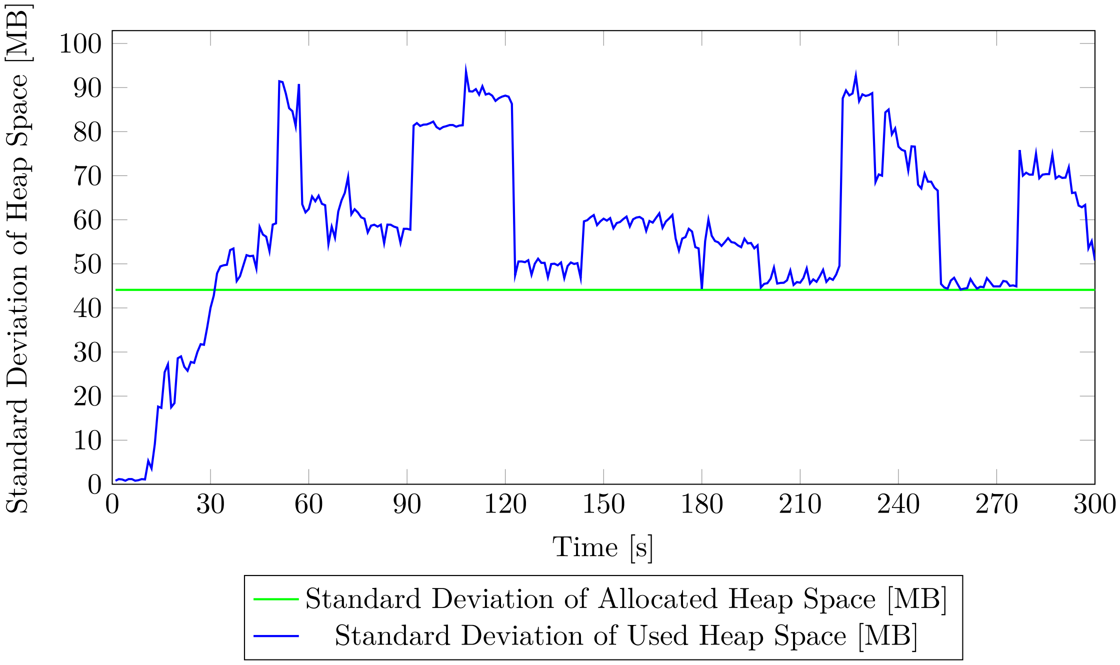


Figure 6.11: The graphs show the progression of the average numbers of live and daemon threads.

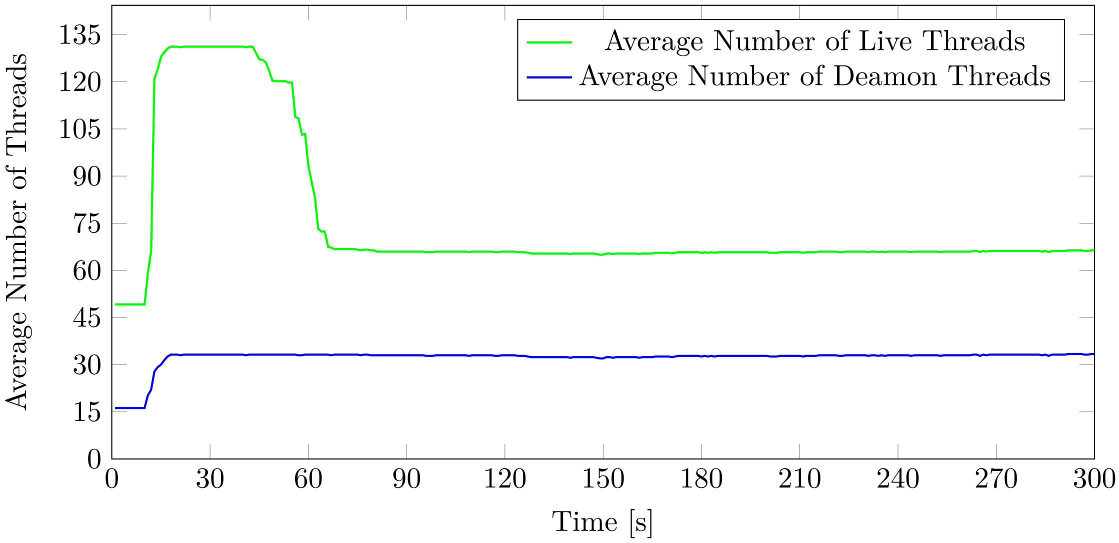
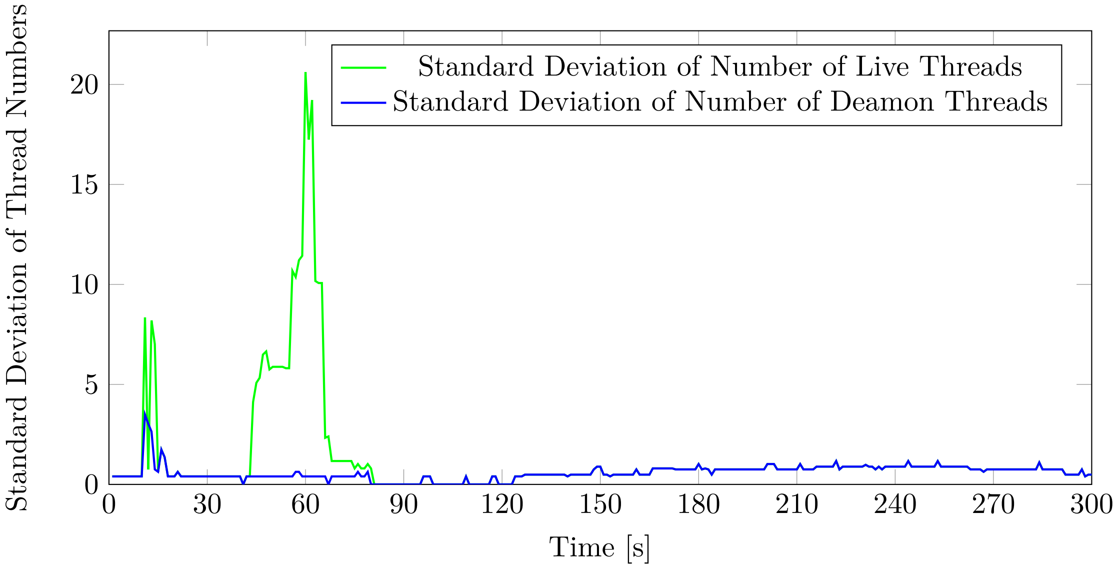


Figure 6.12: The graphs show the standard deviations of the number of live and daemon threads.



6.3.7 Scenario 7: Rapid Decrease of the Network Hash Rate

Description

As outlined in Section 5.2, the Blockchain Manager accepts threshold validation settings that affect the internal threshold validation mechanism. For each metric, a threshold validation function along with a time span can be defined. The time span specifies how long a metric's threshold validation function must continuously evaluate to false until the metric is considered as invalid. The experiment of this scenario analyzes the case when the network hash rate of the currently used blockchain rapidly decreases and violates a defined threshold for the entire time span. Expanse is selected as the currently used blockchain. Since the experiment is conducted with emulated values and the Blockchain Manager's internal logic is analyzed, it can also be performed with Bitcoin, Ethereum or Ethereum Classic without any restrictions. Furthermore, a threshold validation function that returns true if the network hash rate is greater than or equal to 180 GH/s (giga hashes per second) is provided for metric M8 (network hash rate). The time span for metric M8 is set to 15 seconds. The switchover decision function returns true if metric M8 is considered as invalid. The concrete settings are shown in Listing 6.8. The metric M8 is emulated such that the Blockchain Manager is confronted with a rapid decrease of its values. The expected result is that 15 seconds after the first threshold violation has been detected, the Blockchain Manager suggests to switch to another blockchain. One conduction of this experiment is sufficient, since the deterministic functionality of the framework is analyzed and the result is independent from gathered blockchain metrics.

Listing 6.8: Threshold validation settings of Scenario 7.

```
ThresholdValidationSettings validationSettings = new
    ThresholdValidationSettings();
MetricValidationSettings<Double> hashrateSettings =
  new MetricValidationSettings<>(
    hashrate -> hashrate >= (180 * Math.pow(10, 9)), // >= 180 GH/s
    15, ChronoUnit.SECONDS
  );
validationSettings.setNetworkHashrateValidationSettings(hashrateSettings);
validationSettings.setSwitchoverDecisionFn(
  (m1, m2, m3, m4, m5, m6, m7, m8, m9, m10) -> !m8
);
```

Result

Listing 6.9 shows the reaction of the framework in case the hash rate of the Expanse network decreases rapidly. 15 seconds after the first threshold violation has been detected (line 7), a switchover to another blockchain is suggested (line 14). The obtained result matches exactly with the expected one.

Listing 6.9: Log extraction that shows the reaction of the framework in case the network hash rate of Expanse decreases rapidly.

```
13:52:25,189 - Switchover suggestion received: Expanse
13:52:26,983 - Expanse network hash rate: 195.0 GH/s
13:52:31,984 - Expanse network hash rate: 190.0 GH/s
13:52:37,806 - Expanse network hash rate: 185.0 GH/s
13:52:42,807 - Expanse network hash rate: 180.0 GH/s
13:52:46,844 - Expanse network hash rate: 175.0 GH/s
13:52:46,845 - Hash rate (175.0 GH/s) violated
13:52:51,847 - Expanse network hash rate: 170.0 GH/s
13:52:51,847 - Hash rate (170.0 GH/s) violated
13:52:56,852 - Expanse network hash rate: 165.0 GH/s
13:52:56,852 - Hash rate (165.0 GH/s) violated
13:53:01,854 - Expanse network hash rate: 160.0 GH/s
13:53:01,854 - Hash rate (160.0 GH/s) violated
13:53:01,860 - Switchover suggestion received: Ethereum Classic
```

6.3.8 Scenario 8: Rapid Increase of the Costs for Write Operations

Description

This experimental scenario evaluates the situation when the costs for writing data into the currently used blockchain rapidly increase and violate a defined threshold for the entire time span. The time span specifies how long a metric's threshold validation function must continuously evaluate to false until the metric is considered as invalid. Expanse is selected as the currently used blockchain. Since the experiment is conducted with emulated values and the Blockchain Manager's internal logic is analyzed, it can also be performed with Bitcoin, Ethereum or Ethereum Classic without any restrictions. Furthermore, a threshold validation function that returns true if the costs for write operations are lower than or equal to 5 USD is provided for metric M1 (costs for writing one kilobyte of data into the blockchain). The time span for metric M1 is set to 15 seconds. The switchover decision function returns true if metric M1 is considered as invalid. The concrete settings are similar to those illustrated in Listing 6.8. The metric M1 is emulated such that the Blockchain Manager is confronted with a rapid increase of its values. The expected result is that 15 seconds after the first threshold violation has been detected, the Blockchain Manager suggests to switch to another blockchain. One conduction of this experiment is sufficient, since the deterministic functionality of the framework is analyzed and the result is independent from gathered blockchain metrics.

Result

Listing 6.10 shows the reaction of the framework if the costs for writing data into the Expanse blockchain increase rapidly. 15 seconds after the first threshold violation has been detected (line 8), a switchover to another blockchain is suggested (line 15), as expected.

Listing 6.10: Log extraction that shows the reaction of the framework in case the costs for writing data into the Expanse blockchain increase rapidly.

```
15:19:03,471 – Switchover suggestion received: Expanse
15:19:05,915 – Expanse writing costs: 1.0 USD
15:19:10,919 – Expanse writing costs: 2.0 USD
15:19:15,921 – Expanse writing costs: 3.0 USD
15:19:20,925 – Expanse writing costs: 4.0 USD
15:19:25,927 – Expanse writing costs: 5.0 USD
15:19:30,930 – Expanse writing costs: 6.0 USD
15:19:30,931 – Costs (6.0) violated
15:19:35,934 – Expanse writing costs: 7.0 USD
15:19:35,934 – Costs (7.0) violated
15:19:40,935 – Expanse writing costs: 8.0 USD
15:19:40,935 – Costs (8.0) violated
15:19:45,940 – Expanse writing costs: 9.0 USD
15:19:45,940 – Costs (9.0) violated
15:19:45,947 – Switchover suggestion received: Ethereum Classic
```

6.3.9 Scenario 9: Data Movement to another Blockchain

Description

This experimental scenario analyzes the data movement from one blockchain to another during the execution of a switchover. As outlined in Section 5.2, the Blockchain Manager exposes a method for starting a switchover. This method accepts input parameters for specifying the destination blockchain and a date range indicating the amount of data that should be transferred. Due to the low writing costs, we use Ethereum Classic as source and Expanse as destination blockchain. Thus, the Blockchain Manager is configured to select Ethereum Classic as the currently used blockchain. Furthermore, the strings *Master*, *Thesis* and *2018* are written into the Ethereum Classic blockchain through the Blockchain Manager. The date range is defined such that the all data records that have been written into the source blockchain (i.e., Ethereum Classic) during the last hour are moved to the destination blockchain (i.e., Expanse). The expected result is that all three strings are written to the Expanse blockchain during the execution of the switchover. One conduction of this experiment is sufficient, since the deterministic functionality of the framework is analyzed and the result is independent from gathered blockchain metrics.

Result

As shown in Listing 6.11, all three strings have been moved from the Ethereum Classic blockchain to Expanse during the execution of the switchover. Line 5 shows the number of Ethereum Classic transactions that have been already mined (i.e., they are not pending) and have been sent by the address used for the conduction of this experiment. As expected, the framework has retrieved three transactions, one for each test string.

Listing 6.11: Log extraction that shows the movement of data during the execution of a switchover.

```
2018-10-28 19:03:55,490 - start switchover
2018-10-28 19:03:55,493 - Get pending transactions sent by addr
2018-10-28 19:03:55,661 - Pending transactions sent by this addr: 0
2018-10-28 19:03:55,673 - Get mined transactions sent by addr
2018-10-28 19:03:57,973 - Mined transactions sent by addr: 3
2018-10-28 19:03:58,073 - Store data: Master
2018-10-28 19:03:58,216 - Successfully sent transaction:
    0x5282e84ce3af6126dfe047a9f5a5c97e8a3dd79e750adac3606a55a7a9e25a41
2018-10-28 19:03:58,257 - Store data: Thesis
2018-10-28 19:03:58,357 - Successfully sent transaction:
    0x32598dcce61de4bb00b3003c27e8671fde76ece3272c1736cc28f7ec5ea5d8f6
2018-10-28 19:03:58,401 - Store data: 2018
2018-10-28 19:03:58,511 - Successfully sent transaction:
    0x319a2dffce077b2045d575316b764143592e1308477d824eb0ce16bf88636cf9
2018-10-28 19:03:58,511 - switchover finished
```

6.3.10 Scenario 10: Switchover Times

Description

This scenario shows how long it takes to move a specified amount of data from one blockchain to another during the execution of a switchover. Most of the time that is required for a switchover is spent for retrieving data from the source blockchain. Thus, the switchover time strongly depends on this data acquisition process. The goal of this experiment is to illustrate switchover times for diverse amounts of data in order to get a rough overview how long switchovers might take for the specified evaluation setup. For specifying the amount of data that should be moved to the destination blockchain, the Blockchain Manager accepts a time span. For instance, a time span of one hour specifies that all data records which have been mined during the last hour should be moved to the destination chain. The outlined evaluation setup is used without any changes. The Expanse blockchain is used as destination blockchain for all switchover executions, since write operations are very cheap (as outlined in Section 6.3.1). Due to the selection of Expanse as destination blockchain, there is one test case that retrieves data records from Expanse and writes these records back to the same chain (since Expanse is used as source chain as well as destination chain). This special case has no impacts on the results, since the data acquisition process is the same as for other destination blockchains. For conducting this experiment, each blockchain, that is supported by the framework, is used as source blockchain for the switchover executions. Furthermore, the following time spans are tested:

- 12 hours
- 1 day
- 3 days
- 1 week
- 1 month (30 days)

The measurement is conducted five times.

Result

The average values of the measured switchover times are presented in Table 6.7. The corresponding standard deviations are shown in Table 6.8. Since the network nodes have been deployed on the Google Cloud Platform and the framework is running on a local MacBook, it is crucial to note that the framework needs to download relevant information from these nodes in order to extract data that should be moved to the destination blockchain. We performed the evaluation with a download speed of approximately 12 Mbit per second. The concrete switchover times might vary depending on the available download speed. As shown in the table, extracting data from the Bitcoin blockchain takes only a few seconds as opposed to Ethereum, Ethereum Classic and Expanse. These short times can be reached because the Bitcore node stores for each address its unspent transaction outputs and its sent transactions. The Data Access Service responsible for interactions with the Bitcoin blockchain just needs to request these transactions and to iterate over them in order to extract the data. Parity (used for running Ethereum and Ethereum Classic nodes) and Gexp (used for running Expanse nodes) do not support this feature. Thus, the framework must download each block including the full transaction set and it must iterate over each transaction to filter those which have been sent from a particular address. This process is very resource-intensive and time-consuming, since also transactions sent from other addresses must be downloaded and analyzed. Ethereum Classic and Expanse have shorter switchover times than Ethereum, since there are much less transactions stored in these two blockchains.

Table 6.7: The average values of the switchover times measured on 27.10.2018 and 28.10.2018.

Source Blockchain	12 hours	1 day	3 days	1 week	30 days
Bitcoin	1.3 s	1.7 s	1.3 s	2.1 s	2.2 s
Ethereum	4.6 min	7.1 min	19.4 min	57.7 min	3.3 h
Ethereum Classic	16.0 s	54.5 s	2.7 min	6.1 min	25.5 min
Expanse	16.8 s	20.5 s	39.3 s	1.3 s	4.2 min

Table 6.8: The standard deviations of the switchover times measured on 27.10.2018 and 28.10.2018.

Source Blockchain	12 hours	1 day	3 days	1 week	30 days
Bitcoin	0.72 s	1.28 s	0.13 s	0.99 s	1.28 s
Ethereum	33.51 s	53.37 s	5.02 min	9.65 min	24.58 min
Ethereum Classic	1.64 s	0.81 s	4.41 s	9.5	1.34 min
Expanse	1.20 s	0.96 s	2.14 s	2.60 s	5.08 s

6.3.11 Scenario 11: Changing User Preferences and Metrics

Description

This evaluation scenario analyzes the benefit of the framework's selection and switchover mechanism. The experiment is conducted in the context of the motivational scenario outlined in Chapter 4, where multiple organizations exchange information in order to increase their competitive and collaborative advantage. The scenario focuses on an SOA that is made up of different services adopted and operated by several independent and possibly competing business partners. In order to monitor the adherence of SLAs, services publish relevant information to a blockchain. We assume the involved business partners want to use a blockchain that is cheap, fast and has a high level of trust. The framework's Blockchain Manager is configured with the weighted ranking settings outlined in Table 6.9. Since we assume the demand for very cheap and fast write operations, and a high level of trust, we set the weights of M1, M4, M5, M6, M7, M8 and M10 to five (highest importance). The metrics M2 and M3 can be ignored (i.e., we set the corresponding weights to zero), since Bitcoin, Ethereum, Ethereum Classic and Expanse do not charge fees for read operations or for a permanent usage of the blockchain's storage at the time of writing this thesis. We further set the weight of M9 to zero, since in this scenario a comparison of the number of required block confirmations is not expressive. For instance, in Bitcoin a block is assumed to remain in the blockchain with high probability after it is buried under six blocks. With an average inter-block time of about ten minutes, a block is confirmed by six blocks after approximately one hour. Assuming, an Ethereum block should be confirmed by 240 blocks, it would also take about one hour, since Ethereum has an average inter-block time of about 15 seconds. In this example, both blockchains have an average confirmation time of one hour but completely different numbers of required block confirmations. In order to benefit from an accurate selection, we defined the score assignments as granular as possible, e.g., by considering very low costs in the score assignment.

Table 6.9: The weighted ranking settings used for the evaluation of Scenario 11 (metrics with a weight of zero are not listed).

Metric Id	Short description	Weight	Score Assignment
M1	Costs for writing one KB of data	5	$[0; 10^{-4}) \rightarrow 4$ $[10^{-4}; 10^{-2}) \rightarrow 3$ $[10^{-2}; 10^{-1}) \rightarrow 2$ $[10^{-1}; 1) \rightarrow 1$ $[1; \infty) \rightarrow 0$
M4	Exchange rates	5	$[0; 50) \rightarrow 4$ $[50; 100) \rightarrow 3$ $[100; 250) \rightarrow 2$ $[250; 500) \rightarrow 1$ $[500; \infty) \rightarrow 0$
M5	Inter-block time	5	$[0; 20) \rightarrow 4$ $[20; 40) \rightarrow 3$ $[40; 60) \rightarrow 2$ $[60; 120) \rightarrow 1$ $[120; \infty) \rightarrow 0$
M6	Transaction throughput	5	$[10; \infty) \rightarrow 4$ $[5; 10) \rightarrow 3$ $[2; 5) \rightarrow 2$ $[0.45; 2) \rightarrow 1$ $[0; 0.45) \rightarrow 0$
M7	Mining distribution	5	Proportion (%) of the biggest miner: $[0; 22) \rightarrow 4$ $[22; 27) \rightarrow 3$ $[27; 30) \rightarrow 2$ $[30; 38) \rightarrow 1$ $[38; \infty) \rightarrow 0$
M8	Network hash rate	5	Rates are denoted in tera hashes: $[1,000; \infty) \rightarrow 4$ $[700; 1,000) \rightarrow 3$ $[400; 700) \rightarrow 2$ $[100; 400) \rightarrow 1$ $[0; 100) \rightarrow 0$
M10	Reputation	5	$[8; 10] \rightarrow 4$ $[6; 8) \rightarrow 3$ $[4; 6) \rightarrow 2$ $[2; 4) \rightarrow 1$ $[0; 2) \rightarrow 0$

We conduct this evaluation on the basis of the values presented in Section 6.2. Thus, the time line used for this scenario ranges from 25.09.2018 to 17.10.2018. According to their popularity and miner activity, we assume a reputation of ten for Bitcoin and Ethereum, a reputation of nine for Ethereum Classic and a reputation of five for Expanse. In order to emulate an execution on 25.09.2018, the framework is fed with the values measured on this day. According to the weighted ranking settings (which reflects the current user demands) outlined in Table 6.9, we expect Ethereum to be selected as the most beneficial blockchain. The calculation results of the framework are presented in Table 6.10. As expected, Ethereum has been chosen as the most beneficial blockchain. The key points of this selection are as follows:

- By using Ethereum, the costs for writing one KB of data are approximately 24 times lower than the costs that have to be paid for writing the same amount of data into the Bitcoin blockchain. The costs for writing data into the Ethereum Classic and Expanse blockchains are about 154 times and about 96 times lower than the costs that have to be paid in Ethereum.
- Compared to Bitcoin with a price of 6,394.25 USD per coin, the exchange rate of Ethereum is about 30 times lower. The price in USD for one Ether is approximately 20 times greater than the price for one token on Ethereum Classic. With an exchange rate of 0.36 USD, Expanse is the cheapest token.
- Ethereum's inter-block time is about 38 times faster than those of Bitcoin and about 3 times faster than the inter-block time of Expanse. Ethereum Classic has almost the same inter-block time as Ethereum.
- With a transaction throughput of about 5.74, Ethereum processes by far the greatest number of transactions per second, whereas Bitcoin has a rate of 2.57 transactions per second, Ethereum Classic a rate of 0.47 transactions per second and Expanse handles only 0.06 transactions per second.
- The network hash rate of Ethereum is about 16 times greater than those of Ethereum Classic and about 1,275 times greater than the hash rate of Expanse. Bitcoin's hash rate is approximately 217,012 times greater than those of Ethereum.
- The biggest Ethereum miner controls about 24 % of the network's hash power, whereas the biggest Ethereum Classic and Expanse miners control about 42 % and 48 %, respectively. The biggest miner of the Bitcoin network controls about 20 %.

The framework has selected the most beneficial chain in terms of costs, performance and trust, but the selected choice is not always optimal. For instance, Ethereum Classic and Expanse have lower writing costs, but their mining distributions do not satisfy the specified user needs. Thus, the framework has to make a compromise in order to satisfy all user needs at best. Overall, Ethereum satisfies the specified weighted ranking settings best.

Table 6.10: The calculation results for the weighted ranking (based on values measured on 25.09.2018). For each blockchain and each metric, the score and weighted score (denoted as *W. Score*) are shown.

	Bitcoin		**Ethereum**		**Ethereum Classic**		**Expanse**	
Metric	**Score**	**W. Score**	**Score**	**W. Score**	**Score**	**W. Score**	**Score**	**W. Score**
M1	0	0	1	5	3	15	3	15
M2	0	0	0	0	0	0	0	0
M3	0	0	0	0	0	0	0	0
M4	0	0	2	10	4	20	4	20
M5	0	0	4	20	4	20	2	10
M6	2	10	3	15	1	5	0	0
M7	4	20	3	15	0	0	0	0
M8	4	20	1	5	0	0	0	0
M9	0	0	0	0	0	0	0	0
M10	4	20	4	20	4	20	2	10
Total	**14**	**70**	**18**	**90**	**16**	**80**	**11**	**55**

We further assume that engineers of one business partner plan to run data-intensive tests on their adopted services. Since there will be written a huge amount of data into the blockchain, they prefer low costs. Furthermore, these written test data are not used for further processing. The tests just focus on analyzing the write operations. Thus, the level of trust can be neglected. In order to incorporate the changed demands in the weighted ranking system, we set the weights for M7, M8 and M10 to zero. Furthermore, we assume that the settings are changed on 07.10.2018. Due to the lack of significant variations of blockchain metrics between 25.09.2018 and 07.10.2018, Ethereum has been the most beneficial chain for the entire time span. Table 6.11 shows the calculation results based on the changed settings and the metric values gathered on 07.10.2018. Since Ethereum Classic has the highest score, it is considered as the most beneficial chain. Due to a switchover from Ethereum to Ethereum Classic the costs for writing one KB of data

Table 6.11: The calculation results for the weighted ranking (based on values measured on 07.10.2018). For each blockchain and each metric, the score and weighted score (denoted as *W. Score*) are shown.

	Bitcoin		**Ethereum**		**Ethereum Classic**		**Expanse**	
Metric	**Score**	**W. Score**	**Score**	**W. Score**	**Score**	**W. Score**	**Score**	**W. Score**
M1	0	0	2	10	3	15	4	20
M2	0	0	0	0	0	0	0	0
M3	0	0	0	0	0	0	0	0
M4	0	0	2	10	4	20	4	20
M5	0	0	4	20	4	20	2	10
M6	2	10	3	15	1	5	0	0
M7	4	0	3	0	0	0	0	0
M8	4	0	1	0	0	0	0	0
M9	0	0	0	0	0	0	0	0
M10	4	0	4	0	4	0	2	0
Total	**14**	**10**	**19**	**55**	**16**	**60**	**12**	**50**

have been decreased by a factor of 42. Furthermore, Ethereum Classic has almost the same inter-block time as Ethereum (approximately 14 seconds). Since Expanse has an inter-block time of 44 seconds, Ethereum Classic has been preferred.

Furthermore, we assume that an inter-block time between 30 and 60 seconds is completely sufficient for conducting further service tests. Furthermore, the transaction throughput becomes less important for further test executions. Due to the huge amount of test data that is written to the blockchain, low costs have still high priority. We further assume, that the weighted ranking settings are changed on 17.10.2018. Due to the lack of significant variations of blockchain metrics between 07.17.2018 and 17.10.2018, Ethereum Classic has been the most beneficial chain for the entire time span. The score assignment for M5 is changed, such that inter-block times lower than 60 seconds yield a score of

Table 6.12: The calculation results for the weighted ranking (based on values measured on 17.10.2018). For each blockchain and each metric, the score and weighted score (denoted as *W. Score*) are shown.

	Bitcoin		**Ethereum**		**Ethereum Classic**		**Expanse**	
Metric	**Score**	**W. Score**	**Score**	**W. Score**	**Score**	**W. Score**	**Score**	**W. Score**
M1	0	0	2	10	3	15	4	20
M2	0	0	0	0	0	0	0	0
M3	0	0	0	0	0	0	0	0
M4	0	0	2	10	4	20	4	20
M5	0	0	4	20	4	20	4	20
M6	2	6	3	9	1	3	0	0
M7	4	0	3	0	0	0	0	0
M8	4	0	1	0	0	0	0	0
M9	0	0	0	0	0	0	0	0
M10	4	0	4	0	4	0	2	0
Total	**14**	**6**	**19**	**49**	**16**	**58**	**14**	**60**

four. Since the transaction throughput has a lower priority, the weight for M6 is set to three. Based on the metric values measured on 17.10.2018 and the changed settings, the framework selects Expanse as the most beneficial chain, since it has the highest score (as shown in Table 6.12). A switchover from Ethereum Classic to Expanse enables a cost reduction by a factor of approximately 45.

Due to the lack of significant variations of the measured values, we further simulate a rapid increase of the costs for writing one KB of data into the Expanse blockchain, but other values measured on 17.10.2018 remain unchanged. We set the writing costs to two USD. In reaction to this change, the framework selects Ethereum Classic as the most beneficial blockchain (M1 yields a score of zero in the weighted ranking). The switchover from Expanse to Ethereum Classic enables a cost reduction by a factor of approximately 2,231.

To sum up, the assessment of several blockchains, the mechanism for reacting to various events and the switchover functionality enable users to switch to another, more beneficial blockchain in order to benefit from low costs, high performance or better security. The framework is able to react to changed user demands as well as to variations of blockchain metrics.

CHAPTER 7

Conclusion and Future Work

A blockchain is a distributed ledger used by many cryptocurrencies (e.g., Bitcoin or Ethereum) to keep track of transactions and state changes. First blockchain applications have been mainly used to store payment transactions, i.e., value transfers from one party to another. With the advent of Ethereum, the second generation of blockchain applications has become popular. Blockchains of this generation are characterized by their support for smart contracts. The third generation of blockchain applications aims to interact with the physical world. Blockchains of this generation are applicable for the Internet of Things.

Applications of the blockchain technology are still a subject of research. Popular use cases that came up in the last years are digital voting, notary services, SCM, auditing, control of ownership rights, cloud storage and many more.

The suitability of a particular blockchain for a given use case depends on various criteria such as costs for interactions with that blockchain, the time until a data record is permanently included in the blockchain and the distribution of the network's hash power among miners or mining pools. These properties can vary over the time. Thus, a particular blockchain can become unsuitable for a given use case in the future. Such uncertainties can limit the practical value of blockchains.

In order to overcome this limitation, we designed and developed a framework that is capable of switching back and forth between blockchains on the basis of configured settings and measured metrics. The framework can be customized to the desired requirements. It monitors several blockchains, calculates their individual benefits and determines the most beneficial blockchain on the basis of the customization. The reference implementation of the designed framework supports Bitcoin, Ethereum, Ethereum Classic and Expanse.

7.1 Discussion of Research Questions

In Chapter 1, we introduced four concrete research questions that address existing research gaps in the context of blockchain interoperability. We now present concrete answers to these research questions.

RQ1. Which approaches can be used for blockchain interoperability?

The first research question aims to provide an overview of various approaches for establishing interoperability between multiple blockchains. In Chapter 3, we presented related work and state-of-the-art methods in the area of blockchain interoperability and atomic cross-chain swaps. One of the first contributions in this field is the ACCS protocol proposed by TierNolan in 2013. The ACCS protocol allows two or more users of different cryptocurrencies to swap their assets in an atomic and secure fashion. This protocol is also used by projects such as BarterDEX. BarterDEX is a decentralized auction system that enables users to trade cryptocurrencies from one party to another without the need of a trusted third party. Another interesting project is Tesseract, an exchange service that relies on a trusted execution environment. Furthermore, blockchain interoperability can be achieved with the Lightning Network, relays or sidechains. In later sections of Chapter 3, we presented projects that aim to connect multiple blockchains such that a smart contract residing on a particular blockchain can call smart contracts on other blockchains. These projects are the Crowd Machine, Polkadot, Cosmos, Block Collider and Blocknet.

To the best of our knowledge, there are no contributions in the field of runtime selection of blockchains. The current state-of-the-art approaches do not integrate a mechanism for selecting the most beneficial blockchain at runtime and the switchover between blockchains in a single solution.

RQ2. Which blockchain metrics are relevant for the runtime selection algorithm?

The second research question's goal is to provide a definition of expressive criteria for measuring the benefit of blockchains at runtime. In Chapter 5, we presented detailed requirements for the proposed framework. Ten metrics are described, whereas eight of them are automatically gathered by the framework and two of them have to be provided by the framework user. A detailed list of all metrics is given in Section 5.1.

RQ3. How can the runtime selection of an appropriate blockchain and the switchover between blockchains be performed?

This research question aims to provide a mechanism for selecting a suitable blockchain on the basis of gathered blockchain metrics and an approach for the switchover between multiple blockchains. In Chapter 5, we introduced a weighted ranking system that is used by the framework to determine the most beneficial blockchain. The framework user has to assign a weight to each metric that defines the metric's importance. Furthermore, a score function is assigned to each metric. The score function specifies the score that is assigned to a particular metric based on its value. For each metric, the metric's score is multiplied with the corresponding weight. The multiplication results (i.e., the weighted scores) are summed up and the blockchain with the highest sum is considered as the most beneficial chain.

We presented a second mechanism for initiating a switchover. The user can define thresholds and conditions. The conditions dictate whether a switchover is suggested on the basis of detected threshold violations.

In case a switchover to another blockchain is performed, a defined amount of data is moved from the currently used blockchain to the new one. The concrete amount can be specified by the framework user on the basis of the switchover cause (e.g., a particular metric has violated user-defined thresholds).

In Section 5.2, we presented the technical design of the proposed framework. The framework relies on external network nodes that are used to communicate with a blockchain's network, and accesses data through defined interfaces. Therefore, blockchain-specific implementation details are hidden by components that implement these interfaces. Furthermore, the framework is designed to incorporate the reactive programming paradigm, i.e., each new data record affects the entire calculation and blockchain selection algorithm immediately after it has been received by the framework.

RQ4. What are the benefits of using the proposed solution?

The fourth research question aims to provide an evaluation of the proposed framework. In Chapter 6, we reported on the benefits of the framework's selection and switchover mechanism. The evaluation has been conducted in the context of the motivational scenario outlined in Chapter 4. We showed that the framework can achieve significant cost reductions and performance enhancements due to the ability to switch to another blockchain in case of changing user demands or variations of blockchain metrics.

Furthermore, we analyzed the framework in the context of ten further scenarios. We reported on concurrent write operations, data movements between blockchains, concurrent switchovers, write operations with large amounts of data, switchover times, write operations during the execution of a switchover, frequent changes in the blockchain ranking, the framework's resource consumption, rapid decrease of a network's hash rate, and finally on the steadily increase of the costs for writing data into a blockchain.

7.2 Future Work

This section lists possibilities to extend the framework and some of the issues this thesis leaves open for the future.

7.2.1 Adding Support for Further Blockchains

The reference implementation of the proposed framework supports Bitcoin, Ethereum, Ethereum Classic and Expanse. In order to add support for a further blockchain, an implementation of the interfaces *IMetricCollector* and *IDataAccessService* has to be provided. Detailed interface descriptions are outlined in Section 5.2. Furthermore, the new blockchain must be registered at the Blockchain Manager, the central component of the developed framework.

7.2.2 Move Smart Contracts from one Blockchain to another

During a switchover, only data records are moved from the currently used blockchain to another one. In case relevant smart contracts have been deployed to the currently used blockchain, these contracts are not moved to the other chain, since the framework implementation is not capable of moving smart contracts. Such a feature might be relevant if data records are managed by smart contracts rather than stored in transactions. In this case a movement of the data records along with the corresponding smart contract might be a suitable approach.

7.2.3 Tracking of Transactions holding Data Records

The reference implementation stores data records that are written to Ethereum, Ethereum Classic or Expanse in transactions rather than smart contracts. During a switchover, the framework has to download all blocks that have been mined between a particular date range and to iterate over each block's transactions in order to extract the data records which should be moved towards another chain. This process is resource-intensive and time-consuming. A possible solution to improve this process might be a tracking of transactions that hold data records. For instance, this tracking can be a simple data structure that stores for each address the relevant transaction hashes. The tracking mechanism should be integrated into a network node or should be deployed on a publicly available VM, since the data records might need to be available to multiple instances of the framework.

List of Figures

List of Tables

Listings

Acronyms

ACCS atomic cross-chain swap. 8, 25, 27–29

DApp decentralized application. 3

ECDSA Elliptic Curve Digital Signature Algorithm. 13, 21
EVM Ethereum Virtual Machine. 23, 24

HTLC Hashed Time-Locked Contract. 27, 30, 32

IOIS Inter-organizational Information System. 35, 36

SCM Supply Chain Management. 3, 35
SGX Software Guard Extensions. 28
SLA Service Level Agreement. 37
SOA service-oriented architecture. 36, 37
SPV Simplified Payment Verification. 6, 15, 32, 33

TEE trusted execution environment. 28

UTXO Unspent Transaction Output. 16, 20, 24, 30

Bibliography

[acc13] Atomic cross-chain trading. https://en.bitcoin.it/wiki/Atomic_cross-chain_trading, 2013. Last Access: 26.04.2018.

[acc16] ACCT using CLTV - More Effective than a sleeping pill! https://bitcointalk.org/index.php?topic=1340621.0, 2016. Last Access: 26.04.2018.

[alt17] World's first Atomic Swap Wallet from Altcoin.io released. https://blog.altcoin.io/worlds-first-atomic-swap-wallet-from-altcoin-io-released-6f1cfc52d1cc, 2017. Last Access: 26.04.2018.

[Apo17] Rich Apodaca. OP_RETURN and the Future of Bitcoin. https://bitzuma.com/posts/op-return-and-the-future-of-bitcoin/, 2017. Last Access: 28.09.2018.

[ato16] Atomic swaps using cut and choose. https://bitcointalk.org/index.php?topic=1364951, 2016. Last Access: 26.04.2018.

[BCC$^+$13] Engineer Bainomugisha, Andoni Lombide Carreton, Tom van Cutsem, Stijn Mostinckx, and Wolfgang de Meuter. A Survey on Reactive Programming. *ACM Comput. Surv.*, 45(4):52:1–52:34, 2013.

[BCD$^+$14] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. Enabling Blockchain Innovations with Pegged Sidechains. https://blockstream.com/sidechains.pdf, 2014. Last Access: 26.04.2018.

[BG96] Ann Becker and Dan Geiger. Optimization of Pearl's method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem. *Artificial Intelligence*, 83:167–188, 1996.

[BH17] Sean Bowe and Daira Hopwood. Hashed Time-Locked Contract transactions. https://github.com/bitcoin/bips/blob/master/bip-0199.mediawiki, 2017. Last Access: 26.04.2018.

[BHMS05] Rainer Berbner, Oliver Heckmann, Andreas Mauthe, and Ralf Steinmetz. Eine dienstgüte unterstützende webservice-architektur für flexible geschäftsprozesse. *Wirtschaftsinformatik*, 47(4):268–277, 2005.

[bis18] Bisq. https://bisq.network/, 2018. Last Access: 26.04.2018.

[bit17] bitcoin.org. Bitcoin Developer Reference. https://bitcoin.org/en/developer-reference#block-versions, 2017. Last Access: 07.06.2018.

[BJZ+17] Iddo Bentov, Yan Ji, Fan Zhang, Yunqi Li, Xueyuan Zhao, Lorenz Breidenbach, Philip Daian, and Ari Juels. Tesseract: Real-Time Cryptocurrency Exchange using Trusted Hardware. Cryptology ePrint Archive, Report 2017/1153, 2017. https://eprint.iacr.org/2017/1153.
Last Access: 26.04.2018.

[blo18] Blocknet. https://blocknet.co/, 2018. Last Access: 26.04.2018.

[BMC+15a] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*, pages 104–121, May 2015.

[BMC+15b] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*, pages 104–121, 2015.

[Bri18] Adrian Bridgwater. Blockchains Are Verticalizing, So We Need Interoperability. https://www.forbes.com/sites/adrianbridgwater/2018/02/07/blockchains-are-verticalizing-so-we-need-interoperability/#6bed30df7ab9, 2018. Last Access: 05.05.2018.

[btc18] BTC Relay. http://btcrelay.org/, 2018. Last Access: 26.04.2018.

[But16] Vitalik Buterin. Chain interoperability. https://static1.squarespace.com/static/55f73743e4b051cfcc0b02cf/t/5886800ecd0f68de303349b1/1485209617040/Chain+Interoperability.pdf, 2016. Last Access: 26.04.2018.

[Car18] Diane Cardwell. Solar Experiment Lets Neighbors Trade Energy Among Themselves). https://www.statista.com/statistics/377382/bitcoin-market-capitalization/, 2018. Last Access: 08.05.2018.

[CM16] Arlyn Culwick and Dan Metcalf. The Blocknet Design Specification. https://www.blocknet.co/wp-content/uploads/2018/04/whitepaper.pdf, 2016. Last Access: 05.05.2018.

[com14a] Ethereum community. Patricia Tree. https://github.com/ethereum/wiki/wiki/Patricia-Tree, 2014. Last Access: 16.06.2018.

[com14b] Ethereum community. White Paper. https://github.com/ethereum/wiki/wiki/White-Paper, 2014. Last Access: 16.06.2018.

[com16] Ethereum community. Ethereum Glossary. http://ethdocs.org/en/latest/glossary.html, 2016. Last Access: 16.06.2018.

[Com17] Bitcoin Community. Difficulty. https://en.bitcoin.it/wiki/Difficulty, 2017. Last Access: 28.09.2018.

[Com18a] Bitcoin Community. OP_RETURN. https://en.bitcoin.it/wiki/OP_RETURN, 2018. Last Access: 28.09.2018.

[com18b] Ethereum community. Proof of Stake FAQs. https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQs#further-reading, 2018. Last Access: 17.06.2018.

[Com18c] Ethereum Community. The Ethereum Virtual Machine. http://solidity.readthedocs.io/en/v0.4.24/introduction-to-smart-contracts.html#index-6, 2018. Last Access: 22.07.2018.

[Cor18] Chloe Cornish. Growing number of cryptocurrencies spark concerns. https://www.ft.com/content/a6b90a8c-f4b7-11e7-8715-e94187b3017e, 2018. Last Access: 10.05.2018.

[Cos18] Paul Costas. Vitalik Buterin Proposes Data Storage Fees for Ethereum. https://cryptodisrupt.com/vitalik-buterin-proposes-data-storage-fees-for-ethereum/, 2018. Last Access: 11.07.2018.

[CPPRPM14] Julián Chaparro-Peláez, Antonio Pereira-Rama, and Félix José Pascual-Miguel. Inter-organizational information systems adoption for service innovation in building sector. *Journal of Business Research*, 67(5):673 – 679, 2014.

[ct17] Block collider team. Block collider white paper. https://s3.amazonaws.com/blockcollider/blockcollider_wp.pdf, 2017. Last Access: 05.05.2018.

[dec17] On-Chain Atomic Swaps. https://blog.decred.org/2017/09/20/On-Chain-Atomic-Swaps/, 2017. Last Access: 26.04.2018.

[Dou18] Kevin Doubleday. Blockchain for 2018 and Beyond: A (growing) list of blockchain use cases. https://medium.com/fluree/blockchain-for-2018-and-beyond-a-growing-list-of-blockchain-use-cases-37db7c19fb99, 2018. Last Access: 07.05.2018.

[Dzi15] Stefan Dziembowski. Introduction to Cryptocurrencies. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1700–1701, New York, NY, USA, 2015. ACM.

[Fra18] Matthew Frankel. How Many Cryptocurrencies Are There? https://www.fool.com/investing/2018/03/16/how-many-cryptocurrencies-are-there.aspx, 2018. Last Access: 10.05.2018.

[Her18] Maurice Herlihy. Atomic Cross-Chain Swaps. *eprint arXiv:1801.09515*, 2018.

[HF10] Mohammad Kazem Haki and Maia Wentland Forte. Inter-Organizational Information System Architecture: A Service-Oriented Approach. In Luis M. Camarinha-Matos, Xavier Boucher, and Hamideh Afsarmanesh, editors, *Collaborative Networks for a Sustainable World*, pages 642–652, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[htl16] Hashed Timelock Contracts. https://en.bitcoin.it/wiki/Hashed_Timelock_Contracts, 2016. Last Access: 26.04.2018.

[Kas17] Preethi Kasireddy. How does Ethereum work, anyway? https://medium.com/@preethikasireddy/how-does-ethereum-work-anyway-22d1df506369, 2017. Last Access: 16.06.2018.

[KB18] Jae Kwon and Ethan Buchman. Cosmos white paper. https://cosmos.network/resources/whitepaper, 2018. Last Access: 05.05.2018.

[Ken18] Hu Kenneth. Ethereum account. https://medium.com/coinmonks/ethereum-account-212feb9c4154, 2018. Last Access: 16.06.2018.

[Kle18] Jacob Kleinman. Why Bitcoin's Price Is So Volatile. https://lifehacker.com/why-bitcoin-s-price-is-so-volatile-1822143846, 2018. Last Access: 08.05.2018.

[kno11] knowledgesiam. B2B Exchanges & Electronic Hubs. https://knowledgesiam.wordpress.com/2011/10/06/b2b-exchanges-electronic-hubs/, 2011. Last Access: 11.05.2018.

[kom18] Komodo: An Advanced Blockchain Technology, Focused on Freedom. http://beta.phideas.info/en/whitepaper/2018-02-14-Komodo-White-Paper-Full.pdf, 2018. Last Access: 26.04.2018.

[Low17] Janina Lowisz. Cashaa Will Power Financial Transactions For The Zero-Code Blockchain App Development Environment. https://www.reuters.com/brandfeatures/venture-capital/article?id=22110, 2017. Last Access: 05.05.2018.

[LSST06] Jingquan Li, Riyaz Sikora, Michael J. Shaw, and Gek Woo Tan. A strategic analysis of inter organizational information sharing. *Decision Support Systems*, 42(1):251 – 266, 2006.

[Luu17] Loi Luu. PeaceRelay: Connecting the many Ethereum Blockchains. https://medium.com/@loiluu/peacerelay-connecting-the-many-ethereum-blockchains-22605c300ad3, 2017. Last Access: 26.04.2018.

[lyk18] Lykke. https://www.lykke.com/, 2018. Last Access: 26.04.2018.

[map15] mappum. Mercury - Fully trustless cryptocurrency exchange - Looking for testers! https://bitcointalk.org/index.php?topic=946174.0, 2015. Last Access: 26.04.2018.

[Mar18] Jon Martindale. Bitcoin market cap has already dropped more than $80 billion in 2018. https://www.digitaltrends.com/computing/bitcoin-market-cap-80-billion/, 2018. Last Access: 08.05.2018.

[Min17] MindMajix. Bitcoin vs Ethereum vs Blockchain. https://mindmajix.com/bitcoin-vs-ethereum-vs-blockchain, 2017. Last Access: 20.06.2018.

[Mon17] Monetha. How we react to Ethereum's price fluctuations. https://medium.com/@monetha/how-we-react-to-ethereums-price-fluctuations-bb4d00b70cb7, 2017. Last Access: 08.05.2018.

[Mos17] Alex Moskov. Ethereum Classic vs Ethereum: What's the difference? https://coincentral.com/ethereum-classic-vs-ethereum/, 2017. Last Access: 30.06.2018.

[Nak08] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf, 2008. Last Access: 08.05.2018.

[NBF$^+$16] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction.* Princeton University Press, Princeton, NJ, USA, 2016.

[Pat16] Keval Patel. The Ethereum Virtual Machine. https://medium.com/exploring-code/what-is-reactive-programming-da37c1611382, 2016. Last Access: 28.09.2018.

[PD16] Joseph Poon and Thaddeus Dryja. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. https://lightning.network/lightning-network-paper.pdf, 2016. Last Access: 26.04.2018.

[RD05] Florian Rosenberg and Schahram Dustdar. Business rules integration in BPEL - a service-oriented approach. In *Seventh IEEE International Conference on E-Commerce Technology (CEC'05)*, pages 476–479, 2005.

[rep18] Republic Protocol. https://republicprotocol.com/, 2018. Last Access: 26.04.2018.

[Ros17] Ameer Rosic. Ethereum Mining 101: Your Complete Guide. https://www.huffingtonpost.com/entry/ethereum-mining-101-your-complete-guide_us_58b6e1eee4b02f3f81e44e9f?guccounter=1, 2017. Last Access: 17.06.2018.

[SBS18] Christian Schubert, Michael Borkowski, and Stefan Schulte. Trustworthy Detection and Arbitration of SLA Violations in the Cloud (under submission). *7th European Conference on Service-oriented and Cloud Computing*, 2018.

[SHG14] N. Serrano, J. Hernantes, and G. Gallardo. Service-Oriented Architecture and Legacy Systems. *IEEE Software*, 31(5):15–19, Sept 2014.

[Spr18] Craig Sproule. Crowd Machine white paper. https://www.crowdmachine.com/wp-content/uploads/2017/11/Crowd-Machine-Whitepaper.pdf, 2018. Last Access: 05.05.2018.

[Sta16] Elizabeth Stark. What is the Lightning Network and how can it help Bitcoin scale? https://coincenter.org/entry/what-is-the-lightning-network, 2016. Last Access: 26.04.2018.

[sta18] statista. Market capitalization of Bitcoin from 1st quarter 2012 to 1st quarter 2018 (in billion U.S. dollars). https://www.statista.com/statistics/377382/bitcoin-market-capitalization/, 2018. Last Access: 08.05.2018.

[Sun18] Flora Sun. UTXO vs Account/Balance Model. https://medium.com/@sunflora98/utxo-vs-account-balance-model-5e6470f4e0cf, 2018. Last Access: 16.06.2018.

[SWG+17] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Malware Guard Extension: Using SGX to Conceal Cache Attacks. *CoRR*, abs/1702.08719, 2017.

[Tụ18a] Phan Són Tụ́. Data structure in Ethereum | Episode 2: Radix trie and Merkle trie. https://medium.com/coinmonks/data-structure-in-ethereum-episode-2-radix-trie-and-merkle-trie-d941d0bfd69a, 2018. Last Access: 16.06.2018.

[T18b] Phan Són Tụ́. Data structure in Ethereum | Episode 3: Patricia trie. https://medium.com/coinmonks/data-structure-in-ethereum-episode-3-patricia-trie-b7b0ccddd32f, 2018. Last Access: 16.06.2018.

[The18a] TheBlocknet. The Blocknet Protocol: Enabling Blockchain Interoperability. https://medium.com/@theblocknetchannel/the-blocknet-protocol-enabling-blockchain-interoperability-c5766c2165ed, 2018. Last Access: 05.05.2018.

[The18b] Pascal Thellmann. Scalability issues plague Blockchain technology. Meet Zilliqa. https://hackernoon.com/scalability-issues-plague-blockchain-technology-meet-zilliqa-32b57b1228aa, 2018. Last Access: 08.05.2018.

[Tie13] TierNolan. Alt chains and atomic transfers. https://bitcointalk.org/index.php?topic=193281.msg2224949#msg2224949, 2013. Last Access: 26.04.2018.

[Tom17] Matthew Tompkins. FIRST EVER CROSS CHAIN ATOMIC SWAP BETWEEN BITCOIN AND LITECOIN A SUCCESS. http://bitcoinist.com/first-ever-cross-chain-atomic-swap-between-bitcoin-and-litecoin-has-now-taken-place/, 2017. Last Access: 26.04.2018.

[TS16] Florian Tschorsch and Björn Scheuermann. Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies. *IEEE Communications Surveys & Tutorials*, 18:2084–2123, 2016.

[wf17] web3 foundation. Polkadot light paper. https://polkadot.network/Polkadot-lightpaper.pdf, 2017. Last Access: 05.05.2018.

[Woo14] Gavin Wood. ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER. https://ethereum.github.io/yellowpaper/paper.pdf, 2014. Last Access: 16.06.2018.

[Woo17] Gavin Wood. Polkadot white paper. https://github.com/w3f/polkadot-white-paper/blob/master/PolkaDotPaper.pdf, 2017. Last Access: 05.05.2018.

[WV07] Y. Wang and J. Vassileva. A Review on Trust and Reputation for Web Service Selection. In *27th International Conference on Distributed Computing Systems Workshops (ICDCSW'07)*, pages 25–25, 2007.

[xHi15] xHire. Atomic protocol #1. https://www.coincer.org/2015/01/27/atomic-protocol-1/, 2015. Last Access: 26.04.2018.

[ZJ18] Kaiwen Zhang and Hans-Arno Jacobsen. Towards Dependable, Scalable, and Pervasive Distributed Ledgers with Blockchains. Technical report, University of Toronto, 2018.

[ZW17] Taiyang Zhang and Loong Wang. Republic Protocol: A decentralized dark pool exchange providing atomic swaps for Ethereum-based assets and Bitcoin. `https://republicprotocol.github.io/whitepaper/republic-whitepaper.pdf`, 2017. Last Access: 26.04.2018.