



**University of
Zurich**^{UZH}

Design and Implementation of a Policy-based Blockchain Selection Framework

*Daniel Lakic
Zürich, Switzerland
Student ID: 10-728-509*

Supervisor: Bruno Rodrigues, Eder Scheid
Date of Submission: June 6, 2019

Abstract

In den letzten Jahren ist die Beliebtheit von Kryptowährungen stetig gestiegen. Blockchain, die Technologie, die dahintersteckt, bietet zahlreiche Anwendungsmöglichkeiten, die nicht allein auf den finanziellen Sektor begrenzt sind. In der Logistik werden Blockchains unter anderem für die Temperaturüberwachung von medizinischen oder anderen empfindlichen Gütern verwendet. Seit dem Erscheinen von Bitcoin im Jahre 2009 ist eine Vielzahl von Blockchain Implementationen entstanden, welche versprechen, die unterschiedlichsten Anwendungsfälle abzudecken. Diese Spezialisierung führt dazu, dass ein Trade-off zwischen Performanz, Datenschutz und Kosten entsteht. Dieser Trade-off muss für den jeweiligen Anwendungsfall berücksichtigt werden, was eine Recherche über die Stärken und Schwächen einer solchen Implementation erfordert. Um die Entscheidungsfindung einer passenden Blockchain zu erleichtern, stellt diese Arbeit einen Prototyp eines Policy-based Blockchain Selection Frameworks vor, mit Fokus auf die Temperaturüberwachung von Transportgütern. Dieser Prototyp hat die Aufgabe, die Entscheidung für die passende Blockchain mithilfe von vordefinierten Eingabeparametern vorzunehmen.

The popularity of cryptocurrencies has been rising steadily over the last couple of years. Mainly, due to the infrastructure they are based on, called blockchain, which offers numerous applications in different areas beyond the financial sector. One of them is the logistics sector, where it is used for cold-chain temperature monitoring of shipments. Since the introduction of Bitcoin in 2009, new blockchain implementations have been arising, which promise to serve a myriad of purposes. The specialization of these blockchains into different use cases, leads to trade-offs in performance, privacy and costs. This means that choosing the appropriate blockchain implementation for a specified use case, not only requires knowledge about the use case, but also over all the different blockchains and their strengths and weaknesses. This thesis presents a prototype of a Policy-based Blockchain Selection Framework focusing on the use case of cold chain monitoring. The purpose of the prototype is to abstract the specifics of the blockchains, effectively making the decision for the appropriate blockchain based on pre-defined input parameters.

Acknowledgments

I want to thank my supervisors Bruno Rodrigues and Eder Scheid for supporting me throughout this thesis by providing valuable feedback and suggestions whenever needed.

Contents

Abstract	i
Acknowledgments	iii
1 Introduction	1
1.1 Motivation	2
1.2 Description of Work	2
1.3 Thesis Outline	2
2 Related Work	3
2.1 Policy-based Blockchain-agnostic Framework	3
2.1.1 Policy-based Blockchain Selection Framework	4
2.1.2 Blockchain Interoperability API	4
3 Background	5
3.1 Characteristics of a Blockchain	5
3.1.1 Consensus Mechanism	5
3.1.2 Data Accessibility and Writability	6
3.1.3 Blockchain Performance	7
3.1.4 Transaction Costs	8
3.1.5 Smart Contract-based vs. Transactional	8
3.2 Blockchain Overview	9
3.2.1 Bitcoin	9

3.2.2	Ethereum	9
3.2.3	Stellar	10
3.2.4	EOS	10
3.2.5	IOTA	11
3.2.6	Hyperledger Sawtooth	11
3.2.7	Multichain	12
3.2.8	Summary	12
3.3	Policy-based Management (PBM)	12
3.3.1	Policy Conflicts	13
4	Blockchain Selection Framework	15
4.1	Objective	15
4.2	Design of the Solution	17
4.2.1	Architecture	17
4.2.2	Functionality and Data Workflow	18
4.3	User-defined Policies	22
4.3.1	Blockchain-specific Parameters	23
4.3.2	Externally Driven Parameters	24
4.3.3	Policy Parameters Summary	27
4.4	Technology Stack	28
4.4.1	Server and Client	29
4.4.2	Database	30
5	Evaluation and Discussion	31
5.1	Blockchain Selection Evaluation	31
5.1.1	Private Blockchains	32
5.1.2	Public Blockchains	32
5.1.3	Mixed Blockchains and Split Transactions	34
5.2	Discussion	36

<i>CONTENTS</i>	vii
5.2.1 Selection Process	36
5.2.2 Blockchain Characteristics	36
5.2.3 Stability and Security	37
6 Conclusion and Future Work	39
6.1 Future Work	39
Bibliography	41
Abbreviations	49
Glossary	51
List of Figures	52
List of Tables	53
A Installation Guidelines	57
A.1 Installation with Docker	57
A.2 Local Installation (Alternative to Docker)	57
A.3 Configuration	58
A.4 Testing	58
A.5 Run Server	58
A.6 Usage	59
B Contents of the CD	61

Chapter 1

Introduction

The blockchain technology first gained traction as Satoshi Nakamoto introduced a peer-to-peer (P2P) electronic cash system, in his white paper in 2009 [1]. Bitcoin, the name of the cash system, offers a decentralized way to perform monetary transactions, without the need for Trusted Third Parties (TTP). Blockchain, the technology behind Bitcoin, is an open, distributed ledger that can record transactions between two parties efficiently and in a verifiable and permanent way [2]. It relies on cryptographically appended transactions organized in blocks, which once recorded, cannot be altered retroactively. To manage the distributed ledger, the blockchain is operated by a P2P network, which is used for validating new blocks. These properties that the blockchain exhibits, open up a wide range of use cases that can be solved with this technology. Ranging from initially intended uses like cross-border payments, where monetary transfers reduce costs, which would be collected by TTPs as fees [3], up to less obvious use cases, like e-voting, where it allows to identify voters and ensure the validity of the data [4].

Bitcoin's success opened the floodgates for several other blockchain implementations. Each focusing on different aspects and usages. While some are focused on providing high throughput and vertical and horizontal scaling [5], others aim to offer capabilities beyond monetary transactions. For instance smart contract support [6] or blockchain permissions, where not everybody in the chain receives read or write access. With blockchains focusing on these different aspects, it also introduces some trade-offs. For example, while private blockchains do exhibit the benefit of added data security and performance, these benefits come with the cost of the loss of transparency. Private blockchains can shield the chain from the public view, which also enables higher performance as less nodes are used, that need to verify transactions. Though as a consequence, this leads to less transparency and a higher degree of centralization [7].

With these trade-offs in place, there is no question that certain blockchains may fit one use case more than an other. This also explains the meteoric rise of specialized blockchain implementations, promising to solve a myriad of use cases. A blockchain that has fast throughput in terms of Transactions Per Second (TPS), but introduces a certain degree of network centralization is of no use, if the use case asks for secure storage of sensitive data. On the other hand, a blockchain with smart contract capabilities is not necessarily

needed for monetary transfers. Selecting the appropriate blockchain that fits the use case like a glove is of utmost importance.

1.1 Motivation

With a constantly rising pool of available blockchains and with selecting the appropriate implementation being of such high importance, it is difficult to keep up with the specifics the various blockchains exhibit. Thus, finding a way to be able to select blockchains for a specified use case, without having to dig into the specifics of each implementation, would be a way to address said problem. As an example for such a use case, where the blockchain technology is used, this thesis considers the supply-chain area, where goods of all over the world are produced and distributed across a vast network. In this regard, Modum.io AG developed a blockchain-based solution to monitor, amongst other data, temperature data during transport [8]. Since the goods in the supply-chain follow a vast amount of complex processes, different needs for the blockchain technology arise.

1.2 Description of Work

This thesis addresses the issue of selection of a fitting blockchain implementation based on the requirements of a use case. It investigates characteristics blockchain implementations exhibit and explores the area of Policy-based Management (PBM), to propose a prototype for a Policy-based Blockchain Selection Framework. This prototype allows users to define policies, fitting to their use case. Depending on the configured parameters, the policy guides the selection of an appropriate blockchain. Based on a Node.js web service/application, it exposes two different endpoints for users to interact with. On one hand a Graphical User Interface (GUI) developed on top of a REST API is provided for managing policies, on the other hand another REST API endpoint is provided for executing transactions to get recommendations for blockchain implementations, based on the provided data and active policy. Furthermore, an evaluation in terms of the blockchain selection process is conducted, presenting the viability of the solution.

1.3 Thesis Outline

Following this introduction, the related work is presented in Chapter 2, showing how this proposal of a Policy-based Blockchain Selection Framework fits in with existing work. Chapter 3 takes a look at the characteristics blockchains exhibit and the concept of Policy-based Management. This Chapter provides the background for the proposed solution. Chapter 4 introduces the Policy-based Blockchain Selection Framework, providing the objective, explaining the design choices, laying out all the possible configuration options and the used technology stack. Chapter 5 presents the results of the evaluation and discusses the findings. Finally, Chapter 6 draws a conclusion on the viability of the prototype, summarizes the findings and proposes possibilities for future work.

Chapter 2

Related Work

This Chapter introduces the Policy-based Blockchain-agnostic Framework. It presents the core idea behind the framework, its components and how the prototype presented in this thesis fits within this framework.

2.1 Policy-based Blockchain-agnostic Framework

The Policy-based Blockchain-agnostic Framework is a framework, which aspires to connect different blockchains and facilitate their configurations and operation. To achieve this, it aims to provide blockchain interoperability and to offer transparency over operation of different blockchains through Policy-based Management (PBM). This enables users and applications to operate transparently over different blockchains [9].

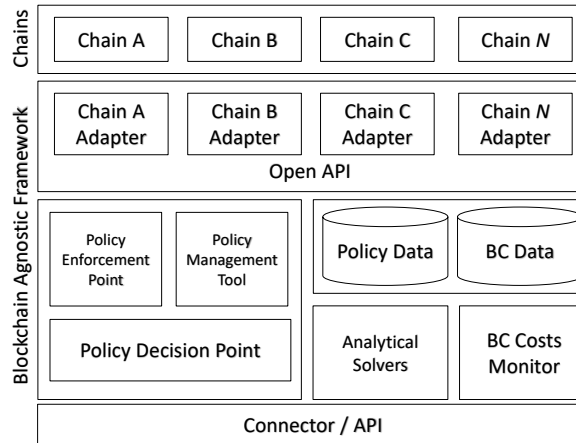


Figure 2.1: Policy-based Blockchain-agnostic Framework architecture [9]

In this framework, the client supporting a certain blockchain is supposed to send a transaction to the framework, which is designed as a server-side component. The framework in turn will transparently convert said transaction to the desired blockchain. Figure 2.1 displays the architecture over the envisioned framework, which is composed of the following components:

- The **Connector / API** signals the point of entry of the framework. It provides an interface for interaction with the framework.
- **PBM** is conducted through an additional component. It is consisting of the **Policy Management Tool** for managing policies, the **Policy Decision Point** for decision over the active policy and the **Policy Enforcement Point** for enforcing the active policy.
- The **Analytical Solvers** calculate usage scenarios based on the user defined restrictions. For example, cost thresholds, selected blockchain implementations and performance measures.
- The **Blockchain Costs Monitor** calculates transaction fees and blockchains rates, which are used for making decisions regarding the user-defined cost thresholds.
- A **Database**, which stores blockchain data and the user-defined policies.
- An **Open API**, which implements adapters to enable cross-chain communication.

The framework is split in to two major components, the Policy-based Blockchain Selection Framework and the Blockchain Interoperability API. Sections 2.1.1 and 2.1.2 will examine the different components.

2.1.1 Policy-based Blockchain Selection Framework

The Policy-based Blockchain Selection Framework is tackled in this thesis. It is the component that enables communication with the framework by letting users configure policies. These policies consist of rules, which the framework uses to decide on the appropriate blockchain implementation. Users are able to configure if they wish to put focus on performant blockchains or aim to minimize costs of the transactions. Depending on these preferences, the framework conducts its decision process. To make these decisions, the framework uses external services to determine costs and has blockchain information stored in its database that contains static blockchain characteristics. When a decision has been made, the result is forwarded to the Interoperability API.

2.1.2 Blockchain Interoperability API

The Blockchain Interoperability API is the component that allows interoperability between the different blockchains. It provides an interface to interact with multiple private and public blockchains in the form of a Python-based API. It contains functions to store and retrieve any arbitrary data on a supported set of blockchains, without having to understand the concepts of the underlying blockchains. The provided API is based on an interoperability technique called “notary scheme” [10]. In a notary scheme a trusted entity or set of entities is used to notify chain X that an event on another chain Y has happened [11].

Chapter 3

Background

Blockchains exhibit a wide variety of characteristics, which makes them more useful for certain use cases and less for others. Selecting the fitting blockchain for a specific use is therefore dependent on the characteristics the blockchains exhibit. This Chapter presents these characteristics and their resulting impact on the operation of the blockchain. In Section 3.2 these characteristics are analyzed on the set of blockchains the prototype and the Interoperability API support. Finally, Section 3.3 introduces Policy-based Management (PBM), which is used as a basis for the prototype.

3.1 Characteristics of a Blockchain

At the core of a blockchain stands the distributed ledger. It is a database held and updated independently by each node in a network. It holds records of transactions, account balances or smart contracts. Records are not communicated to various nodes by a central authority, instead they are constructed by every node. Once consensus in the network has been reached, the distributed ledger is updated [12].

3.1.1 Consensus Mechanism

When nodes perform transactions, a mechanism is needed for keeping track of the account balances and to avoid fraud, such as double spending attacks, by making sure that there is only one consensus ledger. This is called a consensus mechanism, which is used for appending blocks to the ledger that contain transactions [13].

Different consensus mechanisms exist, which each have their own strengths and weaknesses. Proof-of-Work (PoW) was the first consensus protocol proposed by Satoshi Nakamoto. In this protocol, nodes compete to calculate the hash value of the next block. When a node achieves the solution, it waits for the mutual confirmation of other nodes, before the block is added to the existent blockchain. The main disadvantage of this approach is the excessive power consumption [13]. To combat this problem, Proof-of-Stake (PoS)

was proposed. PoS chooses block validators pseudo-randomly, depending on the nodes' wealth and coins at stake [13, 14]. Even though PoS improves upon PoW's power consumption problem, it does not solve the lack of scalability these approaches exhibit. For this purpose, delegated Proof-of-Stake (dPoS) was proposed. It improves the scalability by using a selection process for a set of witnesses. These take turns verifying and creating new blocks. While this approach allows for a scalable solution, it also introduces a higher degree of centralization [14]. Various other implementations of consensus mechanisms exist, each offering a trade-off in performance and centralization.

3.1.2 Data Accessibility and Writability

A blockchain implementation can be put into different categories regarding accessibility and writability. Some definitions split these into three different categories: public, private and consortium blockchains [15, 16, 17]. These sources explain these categories as follows:

- **Public blockchains** are open to the public, meaning everybody can read and send transactions. The network is fully distributed.
- **Private blockchains** are not open to the public, meaning participation in the network is restricted (usually within or between organizations). Not everybody can read or send transactions. This leads to a smaller degree of decentralization.
- **Consortium blockchains** are not fully open to the public, meaning everybody might be able to read transactions, but the sending of transactions is restricted.

Public blockchains allow for complete decentralization and full distribution, as every node has as much transmission power as any other. Every node is treated as an equal. Each needs to authorize a transaction for it to be valid. This allows for complete transparency, which comes with the cost of needed computing power and raises questions about security concerns. Private and consortium blockchains on the other hand operate in a closed or narrower network, which eliminates the concerns caused by complete transparency. This can also be a disadvantage as it (partially) removes the decentralization and distribution aspect of the blockchain [18].

While these sources equate public with permissionless blockchains and private with permissioned blockchains, other sources split these in a further step. These sources equate the labels of private and public to the concept of user authentication (*i.e.*, who the user actually is) and the permissioned and permissionless concept to user authorization (*i.e.*, what the user is able to do) [19, 20]. In this scenario, private and public blockchains are split into permissioned or permissionless blockchains. These distinctions are important when specifying who is granted read and write access and in turn, have an effect on who can control and manage the integrity of the ledger. The following categorization exists [21]:

- **Public permissionless blockchains** grant write/read access to all participants.

- **Public permissioned blockchains** grant read access to all participants, though write access is restricted.
- **Private permissionless blockchains** grant write/read access to all participants within the organization.
- **Private permissioned blockchains** manage write/read access by a centralized organization.

3.1.3 Blockchain Performance

Different factors can be attributed to the performance of a blockchain. The differentiation between public and private plays a role regarding performance as well. Public implementations tend to be slower than private ones, due to their architecture being fully decentralized, as they have a larger number of nodes available in the network, that are participating in the ledger [20]. Another factor is the consensus mechanism used to manage the ledger. The following characteristics exist for measuring the performance of a blockchain solution:

Blockchain Throughput

The blockchain throughput defines the maximum number of transactions, which can be confirmed over a specific period (*e.g.*, max throughput of 3.3 – 7 Transactions Per Second (TPS) in Bitcoin's case) [22]. The amount of TPS a blockchain can handle, is dependent on various factors. The block size determines how many transactions a block can hold. The higher the block size, the more transactions can be processed within a block, increasing the throughput. Other factors include the implemented consensus algorithm, the processing power of the nodes or the frequency of block generation, also known as block time [23].

Block Time

Block time defines the amount of time it will take the network to produce a new block. In order to finalize transactions, new blocks have to be created, so that the transactions can be confirmed by the whole network [24].

Data Size

Data size refers to the amount of data a blockchain is able to store. Transaction-focused blockchains (*e.g.*, Bitcoin) are not designed for storing arbitrary data, while smart contract-based blockchains allow a higher amount. Nonetheless, data can be stored in the blockchain, though some allow larger amounts of data than others [25].

3.1.4 Transaction Costs

Costs generated by transactions can originate from different sources. One of the prominent reasons for costs are transaction fees. In order to successfully execute transactions, new blocks need to be added to the ledger, so that they are verifiable. In PoW-based blockchains, so called miners are needed that confirm the transactions. Confirming transactions is computationally expensive. To give these miners incentives, fees can be included into the transaction [26]. PoW-based blockchains contain rewards for mining outside of fees, which means that theoretically no fees need to be paid, though fees add additional incentives for miners to prefer certain blocks over others. The higher the fees paid by the transaction issuer, the faster a transaction is going to be processed. In the case of Bitcoin, the majority of the earnings of miners is still made up by the mining reward, though once the majority of coins have been mined, the block reward will become a smaller percentage of a miner's earnings. Instead, transaction fees paid by transaction issuers will make up a significantly higher portion of earnings, making transaction fees even more important for the execution of transactions [27].

While transaction fees are one reason for costs that are generated through transactions, another is the computational power used to operate the network. An example for this is the EOS blockchain with its RAM, Network and CPU bandwidth. RAM is used for storage of data into the blockchain. It needs to be purchased up-front before data can be stored. Network and CPU bandwidth are measured as the average consumption over the last three days. They are temporarily consumed, but as the bandwidth decreases, they are reclaimed [28].

Finally, there are also fixed transaction costs. The Stellar network charges a fixed rate, to make sure that the network is not overwhelmed by the amount of transactions [29].

These costs are all generated by public blockchains, since the resources used on the public network need to be paid for. Private blockchain networks do not exhibit these characteristics, as the operator of said network pays for all the resources, which makes them transaction cost free.

3.1.5 Smart Contract-based vs. Transactional

Blockchains like Bitcoin are focused on providing a payment network, that allows to perform transactions between different parties. While this is one of the major uses of blockchains, there are other use cases as well. Ethereum for example aims to provide a platform for decentralized applications. These blockchains take use of a concept called smart contracts, which enable to run computer program code on the blockchain.

A smart contract is a piece of code running on top of a blockchain. It contains a set of rules under which the parties to that smart contract agree to interact with each other [30]. When these rules are met, the agreement is enforced automatically. Smart contracts are triggered by addressing a transaction to the contract. In that case, execution of the smart contract is enforced independently and automatically in a prescribed manner on every

node in the network, taking the data into account that was included in the transaction [31].

While transactional blockchains like Bitcoin, might also have some smart contract capabilities [32], they are very limited. The reason for that is that Bitcoin's script is not Turing complete, since it is not able to solve the halting problem [33]. This restricts the functionality smart contracts of these blockchains can provide.

3.2 Blockchain Overview

This Section provides an overview over the blockchains that are supported by the framework in regards to the characteristics introduced in Section 3.1. These implementations have been chosen to conform with the blockchains the Interoperability API supports.

3.2.1 Bitcoin

Bitcoin, the first decentralized electronic cash system, was introduced by Satoshi Nakamoto to provide a platform for monetary transfers, without relying on a TTP to perform transactions [1].

Due to its consensus mechanism being PoW, it is lacking in performance. It does not reach a high throughput, which is estimated to be around 3.3-7 TPS [22]. The block time of Bitcoin is approximately 10 minutes [34] and it allows a data storage of 80 bytes per transaction [35]. With Bitcoin being a PoW-based blockchain, the transaction costs vary depending on the activity level and the provided fees. Bitcoin is not Turing complete, hence does not offer complex smart contract capabilities [33].

3.2.2 Ethereum

Ethereum markets itself as a “global, open-source platform for decentralized applications” [36]. Hence, it follows a different philosophy than Bitcoin, which is focused on providing a cryptocurrency. For executing code via smart contracts, Ethereum consumes Gas. Gas is the internal unit for computation, which is used for executing transactions or contracts. It can be bought with Ethereum's currency, called Ether [6].

Ethereum offers smart contracts capabilities and is Turing complete, which is why it can store a larger amount of data and offers additional features for providing a platform for decentralized applications. While there is no theoretical limit for transaction sizes, the Block Gas Limit limits the amount of Gas that can be spent for a block [37]. Performance-wise, it suffers from the same issue as Bitcoin, being a PoW-based blockchain, though there are plans to switch to a PoS consensus mechanism [38]. It reaches a throughput of about 15-25 TPS and has a block time of approximately 15 seconds [34, 39, 40]. Transaction fees are determined by the amount of Gas used for the transaction or contract. The Gas price depends on the activity level, as it is used as an incentive for miners.

3.2.3 Stellar

Stellar aims to provide a platform, that moves money quickly, reliably, and at almost no cost. The philosophy is to provide a low-cost financial service for cross-border payments. The name of the currency is Lumens (XLM) [41].

As a result of its own consensus protocol, called Stellar consensus protocol, it can reach high throughput. Depending on the hardware used, it can reach somewhere between 1'000 - 4'000 TPS [42] with a block time of around 5 seconds [43].

The Stellar network requires small fees on transactions and minimum balances on accounts. This is to prevent people from overwhelming the network and to aid in prioritization. The fees consist of the number of operations multiplied by the base fee.

$$transaction\ fee = no.\ of\ operations \times base\ fee$$

The base fee at the time of writing is 100 stroops which is 0.00001 XLM. Writing data into the blockchain consists of a single operation. Stellar requires a minimum account balance to perform operations, which is 0.5 XLM [29].

28 bytes of data can be stored as a “MEMO_TEXT” into the transactions [44]. Since Stellar is focused on providing monetary transactions, it is not Turing complete. Though, it offers limited smart contract capabilities [45].

3.2.4 EOS

EOS serves a similar purpose as Ethereum. Labelling itself as a “blockchain architecture designed to enable vertical and horizontal scaling of decentralized applications” [5].

Compared to Ethereum, it is more efficient, since it uses a combination of dPoS and asynchronous Byzantine Fault Tolerance as its consensus mechanism. This comes at the cost of higher centralization though. EOS runs on only 21 nodes that check and validate new transactions. Every 126 blocks 21 new nodes get elected by preference of votes cast by token holders [5]. This gives a lot of power to these 21 block producers, which contradicts the purpose of a decentralized platform.

However, this smaller degree of centralization enables EOS to reach high performance measures. It has a block time of 0.5 seconds [5] and has reached a max throughput at the time of writing of 3'996 TPS [46]. Though, a study investigated this and reports that a TPS value of 250 was not topped after performing benchmarking with optimal conditions [47]. EOS allows a storage of 256 bytes of data per transaction [48].

While EOS does not have any transaction costs per se, similarly to Ethereum, it uses an internal unit of computation. RAM, the computational unit used for data storage, determines its value by the Bancor Formula for algorithmic pricing. It is an open-source

standard for Smart Tokens, which ensures continuous liquidity and real-time price discovery between blockchain-based assets, without having to match buyers and sellers [49]. The formula for determining the price is as follows:

$$price = \frac{Reserve\ Balance}{Smart\ Token's\ Total\ Supply \times Reserve\ Ratio}$$

Where the Reserve Balance denotes the EOS held in reserve to conduct trading, the Smart Token's Total Supply represents the total amount of outstanding RAM tokens and the Reserve Ratio is the ratio between the Reserve Balance and the Smart Token's total value, which is the market cap of the Smart Token [49, 50].

3.2.5 IOTA

IOTA's vision is to enable device-to-device communication and execute transactions in the internet of things realm [51].

While IOTA is not a blockchain by definition, it is rather similar in concept. Instead of blocks, it uses a directed graph named "tangle" to hold its transactions. The confirmation of transactions happens inside the network by the transaction issuers. When a new transaction joins the tangle, it chooses two previous transactions for confirmation, adding two new edges to the graph. The graph is built with every transaction approving two other transactions. This creates a construct where every transaction can directly or indirectly reference all other transactions as soon as it is confirmed [52].

This unique architecture, allows a throughput of 500 - 800 TPS and the block time is currently fixed at 1 minute [51, 53]. IOTA's maximum data storage size per transaction is 2187 trytes which amounts to 1300 bytes [54]. It does not offer smart contract support, which also means that it is not Turing complete.

3.2.6 Hyperledger Sawtooth

Hyperledger Sawtooth, developed by Intel, is an enterprise blockchain platform for building distributed ledger applications and networks. Compared to the previously mentioned public counterparts, Hyperledger is highly configurable, as it allows to choose the transaction rules, permissions and consensus algorithms, which makes it adaptable to different use cases [55]. Per default the block time is set to 20 seconds and, according to reports, aims to reach a TPS value of 1'000. Though this value is dependent on a myriad of factors, which means that it can exceed or under-perform this value [56, 57].

Since its focus is to provide a platform for blockchains, it has a strong smart contract support in a wide variety of languages [55]. Due to the Interoperability API using IntegerKey transactions to store the data, a maximum value of 20 bytes per transaction can be stored [58]

3.2.7 Multichain

Multichain aims to provide a solution for deployment of the blockchain technology in the financial sector. It solves this by providing control and privacy in a easy-to-use package. It is comparable to Sawtooth in that it also enables to build private blockchains. Multichain is highly configurable and lets the user configure different consensus mechanisms, block sizes and times, permissions and maximum size of the metadata in a transaction, among other configuration options. The throughput is highly dependable on the consensus mechanism of choice [59, 60].

Compared to Sawtooth, it is focused on providing technology for the financial sector, which means it is not focused on complex smart contract capabilities. Thus, it is not Turing complete. Nonetheless, it supports some limited smart contract capabilities for managing dynamic permissions, transferable assets and data storage [61]. The data size limit imposed by the Interoperability API is 80 bytes [35].

3.2.8 Summary

Table 3.1 shows the summary over the blockchains used for this prototype including their key characteristics.

Blockchain	Type	TPS	Time (s)	Data	Turing	Fees
Bitcoin	public	4 - 7	600	80 bytes	No	variable
Ethereum	public	15 - 25	15	46 kbytes	Yes	variable
Stellar	public	1'000 - 4'000	5	28 bytes	No	base fee
EOS	public	250 - 3'996	0.5	256 bytes	Yes	variable
IOTA	public	500 - 800	60	1300 bytes	No	none
Hyperledger	private	variable	20 (default)	20 bytes	Yes	none
Multichain	private	variable	15 (default)	80 bytes	No	none

Table 3.1: Summary of blockchains and characteristics

3.3 Policy-based Management (PBM)

As a basis for this Framework, a PBM-based architecture has been implemented. PBM is a technology for reducing complexity of the task of managing distributed systems or networks. Often used in network management, it allows to define policies, which set certain rules, the system automatically enforces. These rules are defined in Event-Condition-Action format. An example for such a rule for a network, expressed in natural language, would be: “Provide the fastest packet forwarding from 08:00 to 17:00”. When the conditions are met, the policy is activated and the action of using the fastest packet forwarding is enforced, in the event that the system is currently within that time frame.

Policy-based Management consists of the following components, which are also displayed in Figure 3.1 [62]:

- **Policy Management Tool:** Used by the administrator to define and manage policies.
- **Policy Repository:** Defined policies are stored in the Policy Repository.
- **Policy Decision Point:** Retrieves policies from the Policy Repository and interprets them. Makes the decision on which policy to use, which it communicates to the Policy Enforcement Point.
- **Policy Enforcement Point:** Executes the policy, which has been chosen by the Policy Decision Point.

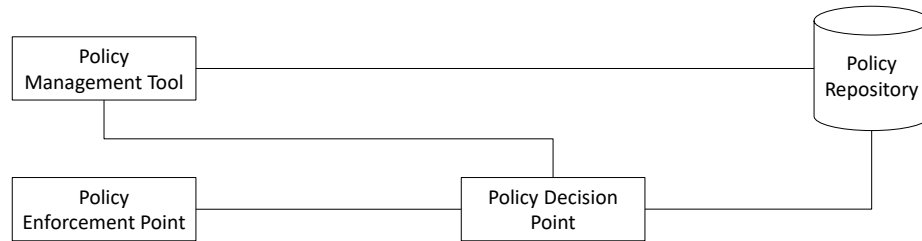


Figure 3.1: Policy-based Management [62]

3.3.1 Policy Conflicts

Due to policies being managed by users, conflicting policies can arise. Policy conflicts emerge, when there are contradictory policy configurations (*e.g.*, two policies with the same activity time frame). As a result, two or more policies are activated which results in contradictory statements [63]. Policy conflicts can arise during two time frames, during policy specification time or run time. They are also known as static and dynamic conflicts.

Policy conflicts that arise during specification time can be eliminated by using static analysis. When a policy is created, a check is conducted to make sure that the currently defined policy does not contain any contradictory statements with one of the already deployed policies. For this case, detection rules can be applied, that detect policy conflicts, should they arise [64].

Dynamic conflicts can only be detected during the enforcement of policies. A domain in which such conflicts are known to arise are resource management policies. These conflicts occur as a result of the current state of the managed system. An example for such conflicting policies are a policy, which increments or decrements allocation of resources and another policy related to setting upper and lower bounds for the resource. While static conflicts can often be managed during policy configuration, dynamic policy conflicts need

to be solved automatically, as to ensure an errorless operation of the system [65]. To resolve these conflicts, either resolution or precedence rules can be introduced [64].

For the prototype presented in this thesis, a static analysis is performed during creation of the policies. Based on defined conflict rules, the framework validates the input to make sure that no conflicts with already declared policies can arise. In case a conflicting policy is about to be defined, the storage of the policy is rejected and the user is prompted to re-configure the parameters, so that no policy conflicts can arise. Also, precedence rules for certain parameters are defined to make sure that always a single policy is active on run time.

Chapter 4

Blockchain Selection Framework

This Chapter explains the prototype for the Policy-based Blockchain Selection Framework developed for this thesis. It starts by defining the objective of the prototype in Section 4.1. Section 4.2 contains a description of what the framework contains, how it is used and how it works. Furthermore, the architecture and interaction with the different components is explained, as well as the configurable policy parameters. Finally, the technology stack used for the prototype is presented and the reasoning behind the choices is revealed.

4.1 Objective

The goal of this prototype is to provide a framework for defining policies, which are used for selection of an appropriate blockchain implementation, depending on the use case that it is needed for. The core problem the framework solves, is that it abstracts the strengths and shortcomings of different blockchain implementations. In turn, it also automates the blockchain selection process. Using a defined set of configurable parameters, a fitting blockchain is selected on transaction execution. The herein introduced framework works well in cooperation with the Blockchain Interoperability API [10]. Using the two frameworks in cooperation, allows for complete automation of the transaction workflow. Starting from selection of the blockchain, up to the execution of the transaction.

During the design and implementation of this prototype, there was an emphasis put on creating a **robust**, **extendable** and **flexible** solution, while providing **user-friendly** set up and usage.

Robustness of the prototype was provided by always ensuring that only one policy at a time can be active, while also restricting the user from creating any policy conflicts. Additionally, the framework ensures, that each defined policy always returns at least a single blockchain implementation, that fits the chosen parameters. Meaning that a consistent output is always to be expected, when API calls are made towards the framework. Finally, during configuration of the policies, the framework informs the user in case some conflicting configurations are about to be made. This effectively eliminates invalid or conflicting policy configurations even before a policy has been created.

```

1 {
2     "_id" : ObjectId("5ca359bcd1297088c8adf971"),
3     "nameShort" : "BTC",
4     "name" : "Bitcoin",
5     "type" : "public",
6     "tps" : 4,
7     "blockTime" : 600,
8     "maxTrxSize" : 80,
9     "turingComplete" : false
10 }
```

Listing 4.1: Example of a blockchain implementation as stored in the database

This prototype supports seven blockchain implementations out-of-the-box, which have been introduced in Section 3.2. Additionally, as a fallback, it also supports a traditional database (PostgreSQL or Postgres [66]) in case no blockchain fits the desired characteristics. Adding additional blockchain implementations is not difficult, as all one would need to do is to add the implementation to the database, including the characteristics (as shown in Listing 4.1) and extend the code with the respective cost model. This makes the framework **extendable** with additional blockchains and features.

While the prototype is configured to work in conjunction with the Blockchain Interoperability API, it is also usable as a standalone application. This makes the proposed solution **flexible** in its usage. In case the framework is used independent of the Blockchain Interoperability API, the user is able to define policies and use the response payload returned by the framework, as a recommendation for which blockchain implementation to use. This is also why the call to the Interoperability API is not made directly after selection and the output is returned instead. Decoupling the two APIs enables a flexible solution that can still be used in conjunction with each other, if one wishes to do so.

Additionally, the framework opens up the possibility to extract information about the transactions. The API used for passing data to the framework returns all necessities for a transaction (as shown in Listing 4.2). This could be used for analysis purposes if, for instance, one wants to compare costs created by using the framework with the regular cost by simply using a single blockchain.

Finally, the GUI the framework uses for managing users and policies, is based on a REST API the framework exposes. Rather than using application-specific interfaces, REST manipulates resources, using a uniform set of “stateless” operations [67]. This means, that the operations performed on the API, which are used for managing the framework, are not dependent on each others state. In turn, it makes using the GUI for management completely optional. This opens up further opportunities for automation, making it possible to manage policies and in turn the whole framework, via external scripts. Interacting with the framework is possible with any programming language, which lets the user choose in what context the framework should be used.

Due to the nature of the framework being a single web application, the setup is **user-friendly**, as it is easier to use than a CLI interface, for instance. Accompanying the

```
1 [
2   {
3     "username": "TestUser",
4     "blockchain": "Ethereum",
5     "dataHash": "hashed data using SHA-256",
6     "data": "the plain data",
7     "cost": 0.01,
8     "policyId": "5ce16ca8fb2adb4b443ae2b9",
9     "costProfile": "performance",
10    "interval": "daily"
11  }
12 ]
```

Listing 4.2: Example of a transaction payload

framework is a docker setup. Docker enables hardware virtualization and isolates applications by using containers [68]. By using the Docker setup, it is only necessary to set the needed environment variables, before the application can be set up with a **docker-compose up** command, which installs the dependencies and populates the database with the necessary information. As for the usage of the framework, a GUI has been created for the management of policies enabling to easily create, edit or delete policies.

4.2 Design of the Solution

The framework provides a web interface to interact with two different components. One, providing a GUI which lets users create, list, edit and delete user policies, while the other provides an API endpoint for executing transactions.

4.2.1 Architecture

The prototype is based on the PBM architecture introduced in Section 3.3. The Policy Management Component allows to configure and manage policies, which are then stored in a database. It is a web service that is accompanied by a GUI, which executes REST API calls in the background. It contains all policies the user has defined, allows to create new policies, delete or edit existing ones and shows statistics about cost and transaction information of the user. The aforementioned statistics are retrieved from the database and served via the REST API. The database contains user information, policies, information about the supported blockchains and all the transactions that have been made. These elements make up the Policy Management Component as depicted in Figure 4.1.

Accompanying the Policy Management Component is the Transaction Component. It is the component used for creating transactions. The user communicates via an API endpoint responsible for recommending blockchains, based on the pre-defined policies, which have been created via the Policy Management Component. When the user makes

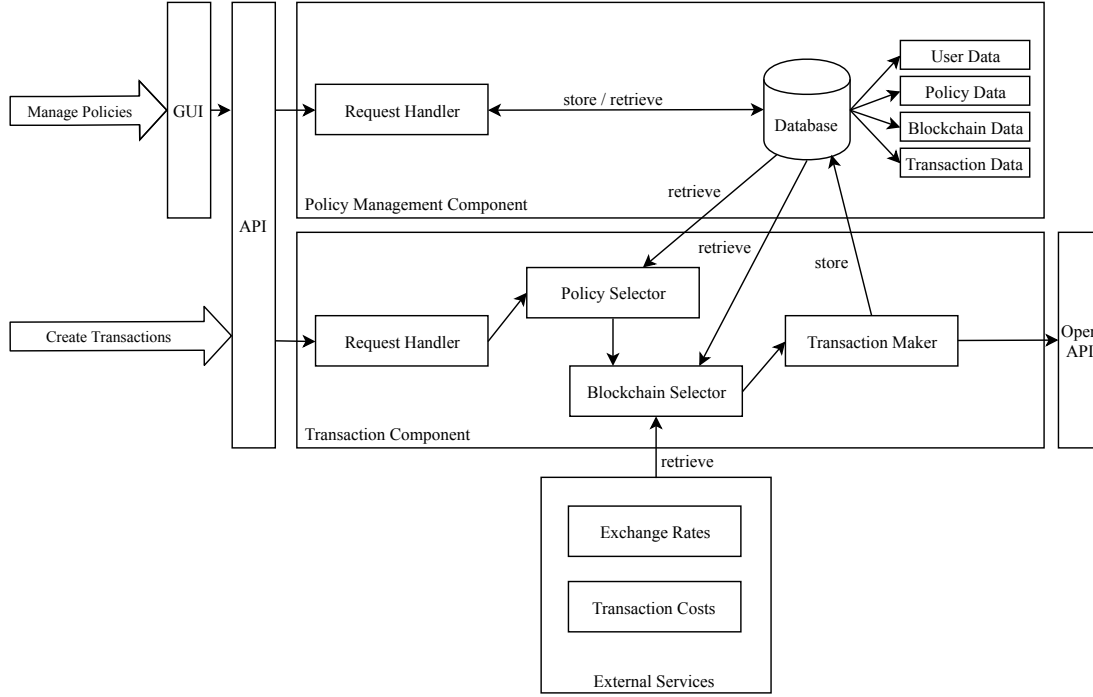


Figure 4.1: Policy-based Blockchain Selection Framework architecture

a call to that endpoint including the data to be stored in the blockchain, the framework executes its selection process to find the appropriate blockchain to be used for the provided data. First, the currently active policy is determined, depending on the parameters which the user has configured beforehand (*e.g.*, cost thresholds). When the active policy has been determined, all blockchains that comply to that policy, are selected, before the result is returned to the user. The Blockchain Selector fetches static blockchain data from the database and uses external services to determine transaction costs of public blockchain implementations. The response of the API call returns the extracted data from the user-provided input, the hashed value of the extracted data, the cost of the transaction, the blockchain implementation chosen for each transaction and additional information used for providing statistics about the transactions. This information can then be used to store the data in the respective blockchain via the Blockchain Interoperability API. Figure 4.1 illustrates the architecture and the dependencies between the components.

4.2.2 Functionality and Data Workflow

This Section focuses on the data workflow of the framework. It provides a detailed explanation on the functionality of the framework and the interaction between the different components introduced in Section 4.2.1. First, the Policy Management Component is examined followed by the Transaction Component.

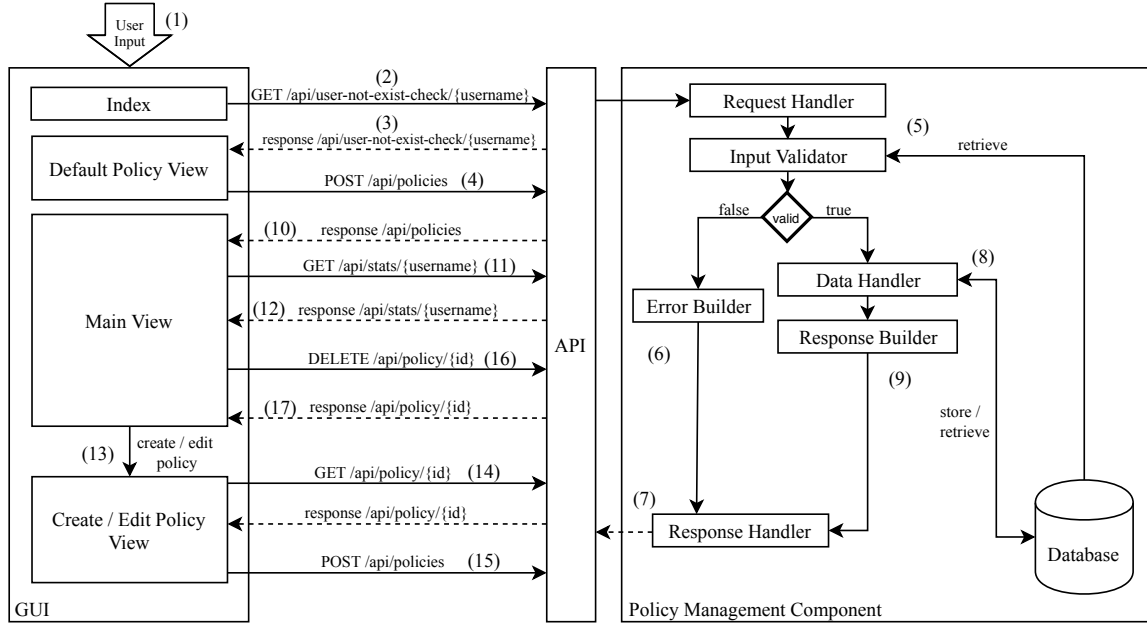


Figure 4.2: Policy Management Component workflow

Policy Management Component

The starting point for the framework represents the Policy Management Component. This component has a GUI build on top of the API that it exposes. When the user enters the index page (1), a username can be entered into the provided form. A submission executes an HTTP GET request to the endpoint `/api/user-not-exist-check/{username}` (2), which is a GET request used to check if a user with the same username already exists. If such a user does exist, the user is prompted with an error message, informing that the username is already taken. If the username is not taken, the user is redirected to a page, which allows configuration of a default policy and the currency of choice (3). Submitting these configurations executes a POST request to the `/api/policies` endpoint containing the username, blockchain characteristics for the default policy and the currency in the request payload (4). The Request Handler of said endpoint processes the data, validates it and then stores the user and the default policy to the database.

Each API request follows the same workflow in the back end. First, the request data is parsed in the Request Handler and validated by the Input Validator, which retrieves data from the database, if necessary (5). In case the input is invalid, the Error Builder builds the error message and error code, which it forwards to the Response Handler (6) before the response is returned (7). In case the input is valid, the data is passed to the Data Handler, which either stores data in the database or retrieves it, depending on the type of request that is executed (8). Following that, the response of the request is built in the Response Builder and passed to the Response Handler (9), which returns the API response (7).

In case the storage of the default policy is successful, the user is then redirected to the main view (10), containing all defined policies, which can be edited or deleted (as depicted in Figure 4.3). Before the user is redirected, the back end fetches all policies for the specified

user and puts them into the view directly. The main view also allows creation of additional policies, shows statistics about the executed transactions so far and shows the currently active policy. The statistics and the policy activity status are fetched via a GET request on the `/api/stats/{username}` endpoint (11). This endpoint is not essential to the operation of the framework, as it simply enables to show statistics and the policy activity in real time, without having to reload the view. The view periodically makes requests to this endpoint to be constantly up to date with the data. When such a request reaches the back end, it assembles the necessary data from the database and formats it in a way which is readable for the front end elements. These are then updated, upon successfully executed request (12).

Policy-based Blockchain Selection Framework

Policy Management

[Create new Policy](#)

No.	Active	Username	BC	Type	Cost	Currency	Inter.	TPS	Time	Size	Compl.	Split	Profile	Time	Edit	Delete
1	Active	User1		indifferent	5	CHF	daily	250	60	20	✓	✗	perf.	08:00 - 17:00	Edit Policy	Delete Policy
2	Inactive	User1	XLM MIOTA	indifferent	10	CHF	daily	250	600	20	✗	✗	econ.	17:01 - 07:59	Edit Policy	Delete Policy
3	Inactive	User1		private	25	CHF	weekly	250	600	20	✗	✗	econ.	24h	Edit Policy	Delete Policy
4	Inactive	User1	ETH XLM EOS MIOTA	public	50	CHF	monthly	20	600	80	✓	✗	perf.	24h	Edit Policy	Delete Policy
5	Inactive	User1		private	75	CHF	monthly	500	600	256	✗	✗	econ.	24h	Edit Policy	Delete Policy
6	Inactive	User1	ETH EOS	public	150	CHF	yearly	20	15	20	✓	✓	perf.	24h	Edit Policy	Delete Policy
7	Inactive	User1	PSG	private		CHF	default	4	600	20	✗	✗	econ.	24h	Edit Policy	Delete Policy

Figure 4.3: Main view of the Policy Management Component

By clicking on a button to create new policies, the user is redirected to a configuration view which contains a form (13). This form allows for creation and configuration of policies. The same view is used for editing policies, in which case the same endpoint to the view is expanded with an additional query parameter, containing the id of the policy (14). In case a policy is edited, the existing data is already filled into the form. As soon as the parameters have been configured and the form has been submitted, a POST request to `/api/policies` is executed, with the request body containing all information about the policy parameters and the username (15). Before the data is stored though, the back end first checks for any policy conflicts (5). If there are no conflicts, the policy is stored in the database (8) and as soon as the front end is notified (7), the user gets redirected to the main view with the updated policies and statistics (10). In case there are conflicts, the user receives an error message informing about the conflict. Conflicts arise in the following cases:

- No blockchain fits the chosen parameters (*e.g.*, public blockchain with a too high TPS threshold).
- A policy with a higher cost interval is not allowed to have a lower cost threshold and vice versa.

- Within the same cost interval and cost profiles, no overlapping time frames are allowed.
- Only one default policy is allowed per user, as it does not contain other distinguishable features, besides the blockchain selection pool.

Finally, in the main view, the user can also delete policies. In case the user clicks the delete button on the corresponding policy, a DELETE request is sent to `/api/policy/{id}` (16). In the back end, it checks whether the policy to be deleted is the default policy (5). If that is the case, the policy does not get deleted and the user gets an error message. This is to make sure that a user always has an active policy. In case it is not a default policy, it gets deleted and the user gets a refresh on the main view (17). Figure 4.2 summarizes the just explained data workflow.

Transaction Component

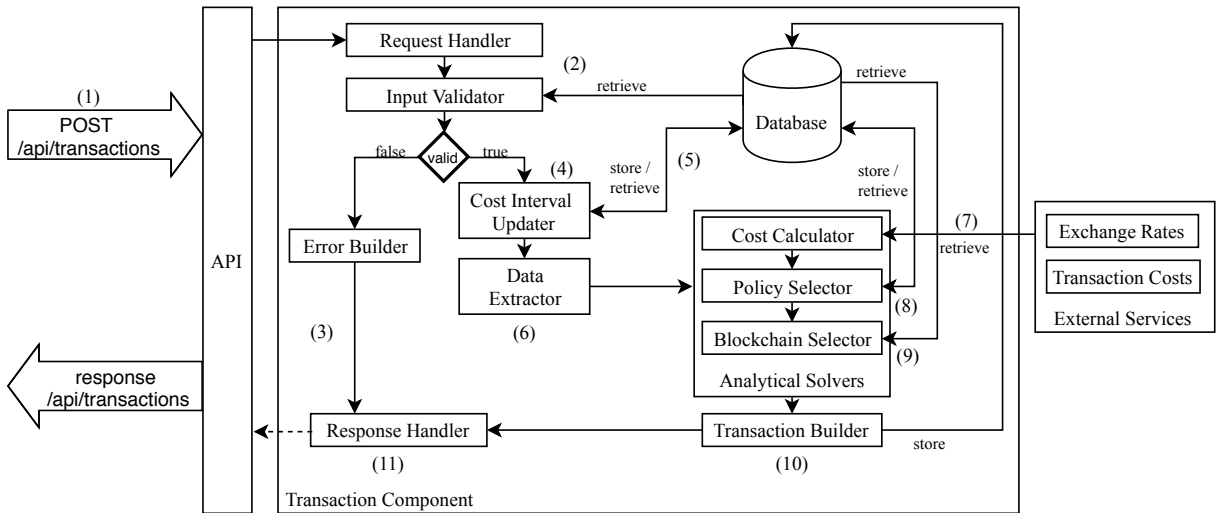


Figure 4.4: Transaction Component workflow

Once policies have been defined, transactions are sent to the `/api/transactions` endpoint (1). To do that, a POST request containing the username, an Excel file containing sheets with temperature data and the minimum and maximum temperature thresholds need to be passed in the body of the request. Alternatively, instead of the Excel file and the temperature thresholds, a string can be passed as well. As soon as this request reaches the back end, the body of the request is checked for valid parameters (2). First, a check that the user has passed all the necessary parameters, is conducted. This means the request body contains a username and either the temperature data and the thresholds or the string to be stored. The request can not contain both a string and a Excel file. This is to not leave any ambiguity about which data should be stored and to make sure to not cause any conflicts. The passed parameters are then further validated. The username must be available in the database and have at least one policy defined and the temperature thresholds must be valid (*i.e.*, minimum temperature lower than maximum temperature). If this is not the case, the API returns an error to the user (3).

If the request conforms to the framework's requirements, the data is further processed. However, in a first step, the system checks whether the accumulated costs over a certain interval have reached said interval (4). This is done by comparing the last update for an interval to the amount of time of that interval (*e.g.*, it checks whether a day has passed for the daily interval, since the last update) (5). If this is the case, the costs for that interval are set to zero.

Following that, the request is processed. In case the Excel file has been sent, sheets are parsed and the violating data is extracted from the sheets (6). Each sheet represents a temperature measurement of a delivery and corresponds to a single transaction. Next, the costs generated by the transactions are fetched (7). For public blockchains, exchange rate data is fetched from CoinMarketCap [69], which is then used to convert the costs, fetched from various external sources [40, 70, 71, 72], to the currency selected by the user. Different calculations take place to convert the costs in to costs per byte, depending on the format the external sources return the respective cost. This is done to limit the amount of requests to the external resources, as it would drastically reduce the performance of the framework, if a call for each sheet would be made. Now, the amount of bytes needed can simply be multiplied by the costs. As soon as those calculations are done, each sheet violation data is examined step by step. First, the policy is selected depending on the cost thresholds (8). The policy with the lowest interval is selected, that is within the time frame and has not reached the max costs yet. From this policy the blockchain selection pool is determined, selecting the blockchains that conform to said policy (9). From the selection pool the most performant or the cheapest one is selected for the transaction, depending on the cost profile the policy exhibits. The calculated costs for the transaction are added to all cost intervals before the next sheet is passed through the same process. In case a transaction split is requested, the framework will split the transactions to send in parallel to multiple blockchains, prioritizing them according the respective cost profile. If the user only passes a string for the transaction, pretty much the same process is conducted, with the difference, that the data is not looped through, instead only one transaction is executed. Finally, the selected blockchains are collected for each transaction (10) and returned to the user (11). This data workflow is summarized in Figure 4.4

4.3 User-defined Policies

Users can define policies, which provide the key component for this framework. These policies represent the set of rules the framework abides by, for selecting the appropriate blockchain, given the state of different factors (*e.g.*, time of the day or cost thresholds). The framework's policy parameters can be split into two different categories. One of them being blockchain specific characteristics and the other being external factors. What follows is an in-depth explanation about parameters that a user can configure for a policy and what kind of consequences these decisions entail.

4.3.1 Blockchain-specific Parameters

These parameters are bound to characteristics the different blockchain implementations exhibit. The characteristics are statically stored in the database as they do not change and are not dependent on any external factors. Hence, the characteristics of the blockchain implementations are mapped one-to-one as choosable parameters for a policy. They are used to filter blockchain implementations based on their characteristics and are not part of the reason why a policy changes. This means that they are responsible for selecting blockchain implementation within a policy.

Public vs. Private

Users may choose whether the data to be stored should be open to the public or not. In case the transactions contain sensitive data, storing into a publicly accessible manner is not an option. When the user makes the choice of whether a public or a private blockchain should be chosen, the framework retrieves data from the database, where the information of whether a blockchain implementation is public or private is stored. In the Policy Management Tool the user can select, whether the blockchain should be public, private or if there is an indifference about the type. During blockchain selection, this information is then used to filter the blockchain selection pool for the active policy.

Blockchain Throughput

The minimum throughput that needs to be supported by the implementation is another choosable parameter. It is labeled as a number, which represents the amount of Transactions Per Second (TPS) the blockchain implementation supports. In case the user needs fast transmissions and has to send out a lot of transactions, the parameter's value can be set to a high number, which in turn eliminates all implementations from the selection pool that do not meet that TPS threshold. In the Policy Management Tool, the parameter's values can be chosen from a list, which contains the values from the respective blockchain implementations.

Block Time

While the blockchain throughput states how many transactions can be made in a second, it does not take in to account how long it takes for the transaction to be confirmed. This is why the block time is choosable for users as well. Similarly to the throughput, in the Policy Management Tool, the value for this parameter can be chosen from a list of available values.

Data Size

Most blockchains are not designed to store a large amount of data. Depending on how much data has to be stored in the transaction, this might be an important factor for the user. In the use case of cold-chain monitoring, the size of the transaction does not matter much, since the data does not get stored into the blockchain directly. Instead, only a hash of the data is stored and the data itself is stored somewhere off-chain. Nonetheless, this is a choosable parameter in case the use case demands more data to be stored in the chain. These size values conform to the the limitations imposed by the Interoperability API. Just as with the previous parameters, the user can choose from the available values and if the value does not meet the threshold, it is removed from the blockchain selection pool.

Turing Completeness

Depending on the use case of the user, Turing completeness of the blockchain implementation is a criteria that might be needed for the blockchain implementation. This is important if smart contracts want to be used in the transactions. As mentioned in Section 3.1.5, even blockchains that are focused on providing transactions can have limited smart contract capabilities, though their lack of Turing completeness limits their complexity. To be able to discern smart contract-focused blockchains, this parameter concerns the Turing completeness and not general smart contract capabilities.

In the provided use case for this thesis, smart contract capabilities are not necessary as the transactions are already verified and just need to be stored in the blockchain. Though, it provides an additional characteristic that differentiates the implementations. For other use cases this might be helpful information. In the Policy Management Tool the user can choose if a Turing complete blockchain is needed or not. Should this be the case, blockchain implementations, which do not have these capabilities, are removed from the selection pool during the selection process.

4.3.2 Externally Driven Parameters

These set of parameters are not static and are susceptible to external factors, for instance the time of the day. They do not map to any blockchain characteristics per se and map user preferences instead. These are the parameters that are responsible for policy changes.

Cost Thresholds

Users can set a maximum amount of costs they are willing to spend in a certain interval, as one of the policy parameters. They can specify any amount as their maximum value in a specified currency (CHF, EUR or USD), they want to spend for their transactions on a daily, weekly, monthly or yearly basis. As soon as the cost within the interval has been reached, the currently active policy switches to the next highest interval. Costs

are calculated by different means. For blockchains that have variable transaction costs, the information is fetched via multiple external APIs that provide information about transaction costs respectively [40, 70, 71, 72]. Since those external sources deliver the fees in the currency of the respective blockchain, an additional call is made to CoinMarketCap [69]. This converts the transaction costs to the previously chosen currency. Since the value of cryptocurrencies is so volatile, this is done with each request to the transaction endpoint of the framework. This makes sure that the costs are always as up to date as possible.

As an example of how this parameter influences the policy selection: When the user provides a maximum cost threshold of CHF 10.- daily and the daily cost has already been reached, the framework switches to the next highest interval or to the economic cost profile of the same interval (this is further explained in Section 4.3.2). If the cost thresholds of all defined policies are reached, the default policy, which does not contain a cost threshold is activated. When defining cost thresholds, the framework always makes sure that the cost threshold of the previous interval does not exceed the threshold of the following intervals. This is to make sure that no policies with a larger interval are rendered useless, as the framework would always skip the next intervals as soon as the threshold for the smaller one has been reached. Before this parameter is used for the blockchain selection within the framework, all the other parameters are used for filtering policy compliant blockchains first. This is done for performance reasons, as this parameter is computationally the most expensive one. As a consequence, the amount of calculations can be limited on the blockchains that fulfill the other parameters. Table 4.1 lists the different blockchains with the respective transaction calculations and their source.

Blockchain	Fee Calculation	Source
Bitcoin	Estimation via APIs (average of 2 sources)	Bitcoinfees [71] Bockcypher [72]
Ethereum	Estimation via APIs (average of 2 sources)	Etherchain [40] Bockcypher [72]
Stellar	Fixed transaction rate: Each operation 0.00001 XLM	-
EOS	RAM price calculation via Bancor Formula	EOS Canada API [70]
IOTA	No transaction fees	-
Hyperledger Sawtooth	No transaction fees	-
Multichain	No transaction fees	-

Table 4.1: Costs of blockchain transactions and calculation methods

Cost Interval

As mentioned in the previous Section, for defining cost thresholds, the user needs to define for which interval this threshold is valid. The framework supports five different intervals. Namely: **daily**, **weekly**, **monthly**, **yearly** and **default**. When an API call

to the Transaction Component of the framework is made, the calculated costs are added to each of the intervals for the current user. These accumulate for each interval until the specified amount of time is reached. As soon as the user interacts with the framework and the time for the interval has passed since the last reset, the costs for this interval are set to zero. This is valid for all intervals except for default. The default interval does not accumulate costs, since it is not bound to costs and represents the fallback, in case all the other policies are currently inactive. Every time the user makes an API call to the transaction component, the default policy ensures that there is always an active policy and a blockchain implementation can be returned. Compared to the other intervals, default does not contain a cost threshold. While policies for the other intervals are optional, every user must have a default policy, which is why the user gets prompted on creation to configure the default policy as a fallback. Only one default Policy per user is allowed, since they do not offer any other distinctive features, besides the blockchain selection pool, which would make it impossible to differentiate between default policies.

Currency

The currency is used for translating cost calculations to a uniform and known entity. As mentioned in a previous Section, the APIs used for calculating costs, always return the cost in the respective blockchain currencies. To be able to calculate costs in a efficient and for the user understandable way, the user is able to chose a currency in which the calculations are done. Just as with the default policy, on user creation, it is necessary to select a currency. Currently, the framework supports **CHF**, **EUR** and **USD** as the choosable currencies.

Cost Profile

Cost profiles provide a way to to put an emphasis on either performance or costs. This parameter is used in case multiple blockchain implementations fulfill the criteria defined by the policy. The users can choose between a **performance** and a **economic** cost profile. In case performance is chosen, the framework chooses the most performant implementations from the blockchain selection pool. Consequently, for economic, the implementation with the lowest transaction costs is chosen from the selection pool. In terms of policy priority, the framework prioritizes performance over economic policies within the same interval. The reasoning behind that, is that an economic profile signals a prioritization in costs. Should the emphasis completely be on reducing costs, no performance policy would be configured for the interval. If there is some priority in performance, a policy with a performance cost profile can be defined, while there can also be a economic one within the same interval. As soon as the cost threshold of the policy with the performance cost profile is reached (*i.e.*, the willingness of the user to spend money on performance), the policy would switch to the economic one.

Transaction Split

Since more than one blockchain implementation can be in the pool of blockchains for a certain policy, a parameter exists to define whether the data to be stored should be spread over all available blockchains or not. In case the transaction split is not chosen, the cost profile chooses a single implementation for all transactions. If a split is chosen, the transactions are spread out over the pool of valid blockchains. The priority is still handled over the cost profile, which means that in case an economic cost profile is chosen, the framework chooses the blockchain with the lowest transaction cost first and vice versa. Consequently, this also means that a not so performant blockchain can be chosen in the performance cost profile for a certain transaction, should the amount of transactions needed for storing all the data line-up with the number of blockchains available in the selection pool. Vice versa for the economic cost profile.

Time Frame

The time frame specifies during which time of the day the policy should be active, given that all the other constraints (*e.g.*, cost thresholds) are fulfilled. This parameter can either be set as valid for the whole day or at a specific time frame of the day (*e.g.*, 08:00 - 17:00). To avoid policy conflicts, for each cost profile within a certain interval, the time frames are not allowed to overlap. For instance, the user has already defined a policy with a daily cost interval and economic cost profile, that is valid between 08:00 - 17:00. Additional policies with the same interval and cost profile, would only be permitted between 17:01 and 07:59. This parameter allows to put emphasis on a certain time frame during the day. For instance, if more performant blockchains are needed during working hours, configurations can be made to emphasise performant implementations during that time frame, while on the other hand less performant implementations can be chosen off-peak.

Preferred Blockchain

Finally, a possibility to select preferred blockchains is also provided. The user can select a single or multiple implementations. This allows users to let the system only choose between blockchains they trust. If the user selects a single implementation, all the other provided parameters are ignored and the policy framework selects the blockchain implementation the user wishes to use. If the user selects multiple implementations, the other parameters are applied on the selected pool of blockchain implementations returning the most suitable one. Should no preferences be chosen, the parameters of the policy are applied to the whole blockchain selection pool.

4.3.3 Policy Parameters Summary

Table 4.2 summarizes all parameters that can be defined in a user policy including the values that can be chosen.

Parameter	Values	Description
Preferred Blockchains	Bitcoin Ethereum Stellar EOS IOTA Hyperledger Multichain Postgres	The blockchain implementations. These values represent the blockchains supported by the Framework. The user can select none, one or multiple.
Blockchain Type	public private indifferent	The blockchain types. These values ask the user if there is a preference over which type of blockchain should be used to store the data.
Max. Cost	Number (up to two decimal digits)	The maximum cost the user wishes to spend for the policy during the specified interval.
Currency	CHF EUR USD	The currency, the calculations are done in. This parameter is mapped to the policy, but is only choosable at the beginning and valid throughout all policies.
Cost Interval	daily weekly monthly yearly	The interval, within which the Max. Cost is valid.
Min. Supported TPS	TPS values	The minimum supported throughput (in TPS) the user needs.
Max. Block Time	Block Time values	The maximum amount of time a block takes to be created.
Min. Supported Data Size	Data Size values	The minimum supported transaction size the user needs.
Turing Completeness	Yes No	Whether the implementation needs to support Turing completeness.
Split Transactions	Yes No	Whether transactions should be split over different blockchains.
Cost Profile	performance economic	Whether the user wants to emphasise usage of performant or economic blockchains.
Time Frame	Time (from-to)	The time frame during which the policy is valid.

Table 4.2: Choosable parameters for a policy

4.4 Technology Stack

The framework at its core is a web application. The reasoning for choosing to build the prototype based on a web application is, that it allows to combine GUI-based components and APIs really well. As mentioned in Section 4.1, one of the points of emphasis is to create an application which does provide a user-friendly interface, but on the other hand

can also be flexible enough to be used in an automated context. HTTP-based REST APIs allow to combine these two worlds really well, which is the main reasoning this architecture has been chosen. This Section provides an overview over the technology stack used to provide the web application.

4.4.1 Server and Client

The core of the framework is based on a Node.js server application. Node.js is an open-source, cross-platform JavaScript (JS) run time environment, that executes JS code outside of a browser [73]. It is based on the V8 Engine, which is a C++ based high-performance JS and WebAssembly engine. V8 has initially been developed by The Chromium Project for Google Chrome and Chromium web browsers [74]. It compiles the JS code to native machine code, instead of interpreting it as bytecode as is more common for this use case, which is where Node.js gets its speed from [75]. Other tasks it handles is the call stack of JS functions, management of the memory allocation for objects, garbage collection and it is responsible for providing all the necessary data types, operators, objects and functions [76]. To run its asynchronous I/O operations Node.js relies on a single-threaded event loop, which is provided by the Libuv library. It is a multi-platform support library with a focus on asynchronous I/O written in C [77]. By providing the event loop, Libuv allows Node.js to perform non-blocking I/O operations, despite the fact that JS is single-threaded [78].

By compiling JS to machine native code, Node.js allows usage of JS on the server side. This makes it possible to write full-stack JS web applications, which are performant, highly responsive as well as easily scalable thanks to the asynchronous nature of Node.js [75]. By being able to write JS on the server side as well as on the client side, a developer has to only rely on one language to create a web application. All these reasons made the choice to use Node.js for this prototype an easy one.

For this prototype, Node.js provides the server of the various components. To handle HTTP requests the prototype uses the often accompanied Express Framework. Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications [79]. Main features of Express include the addition of simple to implement routing capabilities, query string, path string and response body parsing, grace full error handling and support for connect middleware, which allows to extend the application with additional plugins. These capabilities helped tremendously during the creation of the REST API, as they allowed to handle requests and the request data in a simple and extendable way. Express also allows to configure various templating engines for delivering HTML files. For this prototype, Nunjucks has been used to render the HTML used for the GUI. Nunjucks is a templating language specifically made for JS, which is developed by Mozilla [80]. It allows to dynamically pass data to HTML files based on the data passed from the rendering function. This helps eliminate duplication of HTML snippets and allows reusability of often used components.

4.4.2 Database

The REST API provided by the Node.js/Express server for managing users and policies, is designed to perform CRUD operations throughout the application. CRUD is an acronym the four basic functions of persistent storage, namely Create, Read, Update and Delete [81]. CRUD-based applications apply the paradigm throughout the different layers of the application. At the user interface level, they allow interaction with the app to create, list, edit, and delete data, as it is the case for the policies in this prototype. At the HTTP layer, CRUD operations are mapped via the different HTTP methods, namely POST, GET, POST/PUT/PATCH, DELETE. Finally, at the database layer it is used for storing, reading, updating and deleting entries. At the core of these operations stands the database. The prototype uses MongoDB as its database storage, which is a document-based, NoSQL storage [82]. Compared to conventional relational databases, MongoDB does not use SQL to perform operations on data and data is not stored in relational tables. Additionally, while SQL databases conform to a predefined Schema, NoSQL databases allow dynamic structures of the data, which means that each document can have its own structure [83]. MongoDB stores data in JSON format (as presented in Listings 4.1 and 4.2) in a document-based fashion. Each document is identified by a unique ObjectId, which can be used for retrieving a specific document. The decision to use MongoDB has been made, as the policy and blockchain data fit really well into a document-based store, since they can be treated as single entities and do not require any joins. As for the policies, retrieving a policy retrieves all information it contains, without the need for joining different tables. The NoSQL approach also allows for scalability and easy adaptability, should the framework evolve in the future. For example, if new characteristics are added to the blockchain implementations or additional parameters are added to the policy, backwards compatibility is ensured without the need for adapting previous documents. Finally, since MongoDB is storing data in JSON, it integrates well with a JS-based framework like Node.js. By using MongoDB in combination with Node.js applications, serialization and deserialization are automatically taken care of, since the documents are already stored in JSON format.

Chapter 5

Evaluation and Discussion

This Chapter analyzes the prototype in regards to the blockchain selection process. Following the explanation of the methodology and the presentation of the results, a discussion of the findings in regards to the selection process, the blockchain characteristics and the stability and security of the framework as a whole, is performed.

5.1 Blockchain Selection Evaluation

For evaluation of blockchain selections, three scenarios have been conducted. In these scenarios 1'000 requests have been sent to the Transaction Component via POST request to the `/api/transactions` endpoint. Each request contains a Excel file consisting of 10 sheets, which amounts to 10'000 randomly generated transactions. For the performance data of the public blockchains (*e.g.*, TPS values), the lower spectrum of the reported values have been chosen. As for the private ones, values from the performance evaluation of the Interoperability API have been chosen. The evaluation focuses on the parameters introduced in Section 4.3.2 and only changes the blockchain selection pool by setting preferred blockchains. This is because the other parameters only influence the blockchain selection pool and do not cause a change in policy.

To achieve the whole set up, Mocha in combination with the library Supertest has been used. Mocha is a JS test framework for Node.js running tests serially, which allows flexible and accurate reporting of test cases [84]. Supertest provides abstraction for testing HTTP requests [85], which comes in handy for testing the API request, that executes the transactions. For illustrating the outcome of these scenarios, Chart.js has been used. Chart.js is a JS frontend library for plotting charts [86]. These charts are fetching data from the `/api/stats/{username}` endpoint and plot various information about these scenarios. The accumulated costs for each interval compared to the threshold of each interval, is illustrated on the first chart. In the second chart, the cost threshold of a single policy is plotted against the cost spent on each policy. The third chart shows the amount of transactions done on each policy and the last two charts represent the amount of transactions for each blockchain per cost profile and interval respectively. Before the tests can be conducted, policies have to be defined. These are listed in each scenario.

5.1.1 Private Blockchains

For the first scenario, policies containing only private blockchains have been chosen. Table 5.1 shows the defined policies for the user. The policies have been chosen in a way that the two private blockchains and the traditional database, each get compared to each other at least once, starting on the first daily policy and ending on the yearly policy with the whole selection pool. As for the default policy, Postgres has been chosen as a single implementation.

No.	BC Pool	Interval	BC Type	Cost (CHF)	Profile
1	All private BC	daily	private	5	performance
2	HYP, MLC	daily	private	8	economic
3	MLC, PSG	weekly	private	20	performance
4	HYP, MLC	monthly	private	50	performance
5	All private BC	yearly	private	100	economic
6	PSG	default	-	-	-

Table 5.1: Policies for the evaluation of private blockchains

When choosing this configuration, a trivial result happens. Since none of the private blockchains have transaction costs associated with them and the policies do not have an activity time frame set, policy number one stays active forever and Postgres gets chosen for all transactions. This is because the accumulated costs do not rise and since the cost threshold of the policy is never reached, the active policy does not change. In costs, Postgres is tied with the others and offers the best performance, since it is not a blockchain. Thus, it is chosen for all requests.

To solve this issue, a base cost is added to each blockchain implementation (including Postgres). Each transaction contains a base cost for financing infrastructure used for private blockchains or other costs generated by the company operating the framework. This is also the case for all future scenarios, as the public blockchain IOTA does not have any transaction costs either. Running the test a second time, while adding an arbitrary base cost of CHF 0.01 for each transaction, yields a more interesting result.

As depicted in Figure 5.1 Postgres is still chosen for most requests, even though the default policy is never reached. Also, Hyperledger is chosen for none. The reason for that is, that all three possible choices for storage offer the same cost, while Hyperledger offers the lowest performance, according to the blockchain characteristics used for this evaluation. Hence, the framework is always going to prefer the other two solutions. When all three are compared to each other, the same scenario occurs as in the first run, Postgres wins out in performance over both, while Multichain wins out over Hyperledger.

5.1.2 Public Blockchains

For this scenario, policies containing only public blockchains have been chosen. Table 5.2 shows the configured policies for the user. The configurations have been chosen in a way,

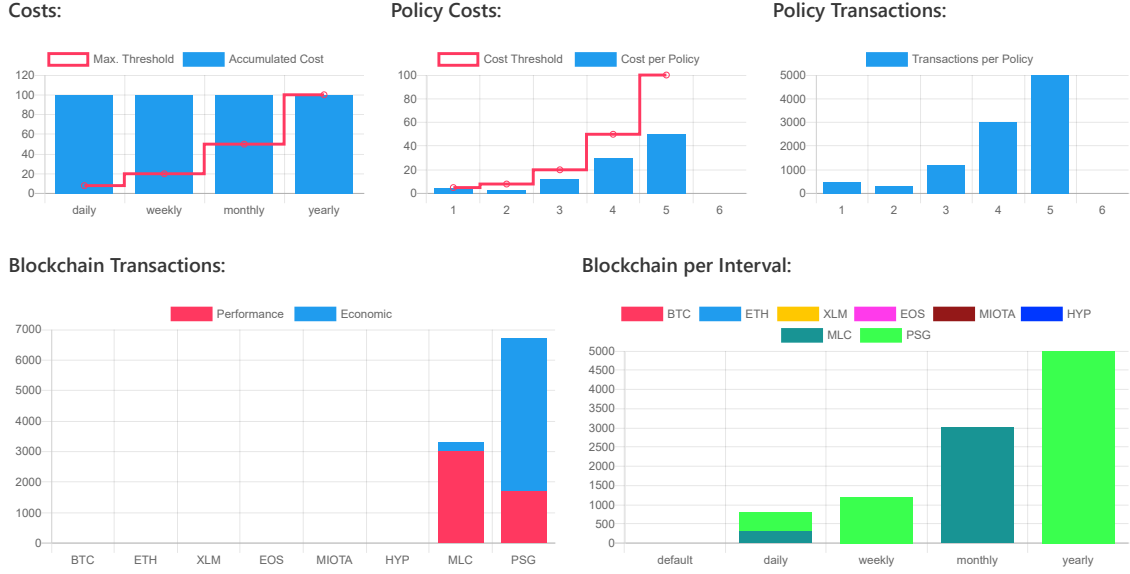


Figure 5.1: Metrics of the evaluation of private blockchains

so that each blockchain implementation is pinned against each other in each cost profile. Finally, the whole blockchain pool is compared for the performance and the economic cost profile once. The default policy is set to Ethereum.

No.	BC Pool	Interval	BC Type	Cost (CHF)	Profile
1	All public BC	daily	public	2	performance
2	BTC, ETH, EOS	daily	public	10	economic
3	EOS, IOTA, XLM	weekly	public	15	performance
4	BTC, EOS	weekly	public	30	economic
5	ETH, EOS, IOTA	monthly	public	40	performance
6	BTC, EOS, XLM	monthly	public	60	economic
7	ETH, EOS	yearly	public	80	performance
8	All public BC	yearly	public	100	economic
9	ETH	default	-	-	-

Table 5.2: Policies for the evaluation of public blockchains

Investigating the outcome of this configuration shown in Figure 5.2, one can see that three implementations dominate the selection. Two of them, Stellar and IOTA, for obvious reasons. Both are relatively cheap in transaction costs and offer the best and second best performance, according to the values used in this evaluation. Ethereum, the third dominating one, has this many transactions since it is the only implementation in the default policy. Looking at the transaction costs, one can see that compared to the evaluation of the private blockchains, the costs far exceed the defined threshold for the yearly policy. This is mostly because the transaction costs of Ethereum are more expensive than those of Stellar or IOTA, by a wide margin. The Policy Costs chart illustrates this well. The costs spent on policy number nine, which is the default policy, are astronomically higher than for any other policy. Precisely, while policies one through eight have an accumulated cost of approximately CHF 100.-, policy number nine alone more than quadruples that

value.

This can also be observed in the two daily policies. While policy number one only contains a cost threshold of CHF 2.-, which is one fifth of the cost threshold for policy number two, far more transactions are made to Stellar (which is the chosen implementation for policy one), than to Ethereum, which beats out EOS and Bitcoin in transaction costs in policy number two. Similar behaviour can be observed in the weekly interval. For policy number three, Stellar beats out EOS and IOTA in performance and about ten times the transactions of policy number four are made, even though policy number four has a three times as high threshold. Policies five and six contain the highest number of transactions, which is because the two cheapest implementations win out the selection pool. Finally, for the yearly intervals, a small amount of transactions is made to EOS, since it beats out Ethereum in performance, but soon enough, because of the high transaction costs, the policy switches and IOTA is chosen, as it is the cheapest one. As soon as the cost threshold is reached here as well, the policy switches to the default policy, which is where the costs explode. Bitcoin is never chosen from the selection pool, which is no surprise, as it offers the worst performance and has the highest transaction costs.

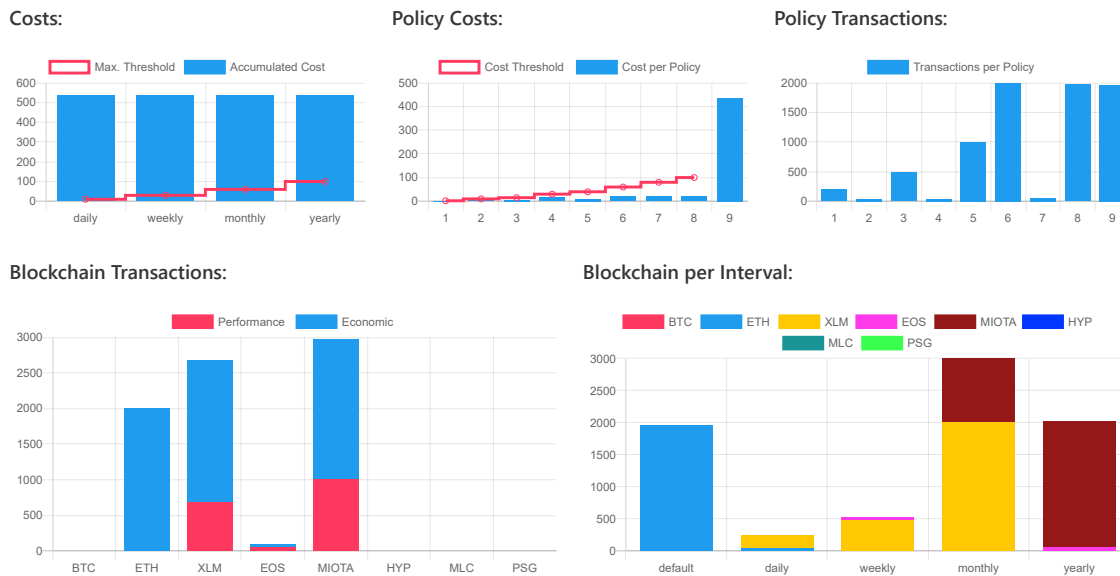


Figure 5.2: Metrics of the evaluation of public blockchains

5.1.3 Mixed Blockchains and Split Transactions

So far, the public and private blockchains have been handled separately. For the final scenario the blockchain type is going to be mixed up, though with the additional tweak of split transactions. The reason for adding the split is, that a simple selection would not fetch a surprising result. Private blockchains offer the better performance and the better cost compared to the public ones. Thus, as soon as one of the private blockchains are in the blockchain selection pool, they would get chosen. And if the selection pool contains only public or private ones, the same behaviour will be observed as in the previous two scenarios. Table 5.3 contains the policy configuration for this scenario.

No.	BC Pool	Interval	BC Type	Cost (CHF)	Profile	Split
1	All private BC	daily	private	15	performance	✓
2	All private BC	daily	private	30	economic	✓
3	All BC	weekly	indifferent	1500	performance	✓
4	All public BC	monthly	public	4000	performance	✓
5	All public BC	monthly	public	8000	economic	✓
6	All BC	yearly	indifferent	15000	economic	✓
7	PSG	default	-	-	-	-

Table 5.3: Policies for the evaluation of mixed blockchains including split transactions

The configuration has been set up, so that each combination of cost profiles and blockchain types is executed once (*i.e.*, all blockchains, all private blockchains and all public blockchains each twice for the respective cost profile). What can be observed, compared to the previous scenarios, is that the cost threshold suddenly jumps from CHF 30.- up to CHF 1'500.- from the daily to the weekly policy and the yearly policy ending at a cap of CHF 15'000.-. Investigating the results illustrated in Figure 5.3 tells the story why this is the case.

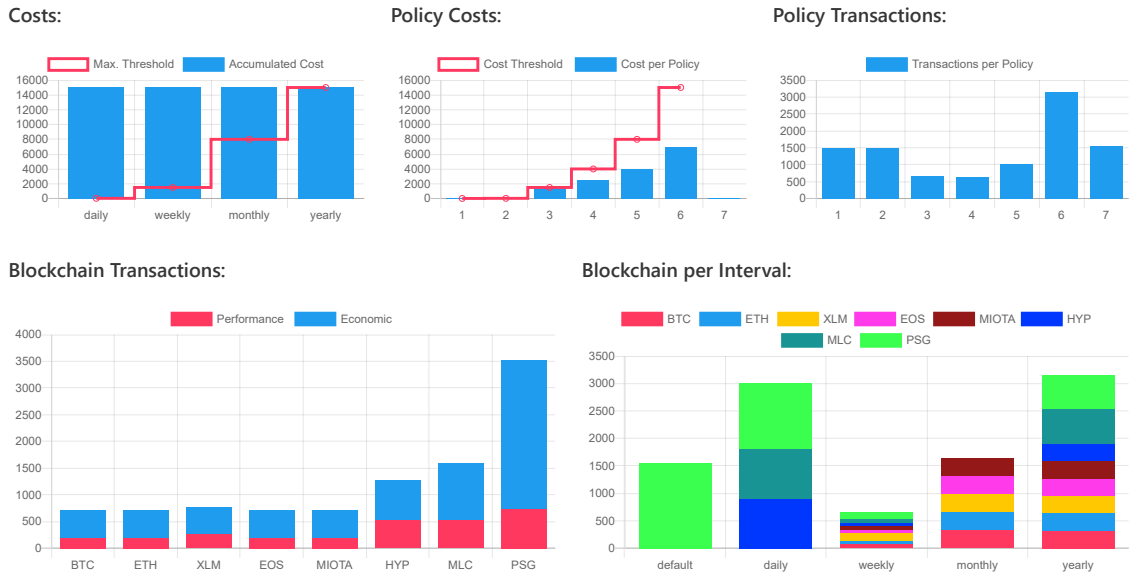


Figure 5.3: Metrics of the evaluation of mixed blockchains including split transactions

The amount of transactions for the daily interval almost equals the transactions for the yearly interval, even though the cost threshold for the daily policies caps out at CHF 30.- compared to CHF 15'000.- for the yearly ones (or more accurately: CHF 7'000, if the previous costs are deducted). This phenomenon is caused particularly by Bitcoin, but also other public blockchains, having astronomically higher transaction costs than the CHF 0.01 for the transaction cost free private blockchains. As soon as public blockchains enter the pool of selectable blockchain implementations, the costs rise exponentially. Based on the blockchains chosen for the transactions, one can see that the private blockchains have been chosen more often, with Postgres having the highest number of transactions (this is skewed by Postgres being the default policy). This behaviour is not surprising. The

transactions have ten sheets each, which means that ten transactions have to be split over the blockchain pool, which contains only eight possible implementations. In this case the framework prioritizes the split depending on costs or performance, which means after the blockchain pool has been exhausted, the split starts at the most performant or cheapest one again, going up to the least performant or most expensive one, depending on the chosen cost profile. This means, that when the whole blockchain pool is chosen, the private ones are prioritized, given their higher levels of confidentiality and performance compared to their public counterparts. However, the trade-off lies in the degree of centralization, in which private ones typically rely on trusted nodes to distribute transactions. When only the private pool is chosen the same behaviour can be observed as in the private blockchain scenario. With all private implementations having the same cost, the most performant is always prioritized (*i.e.*, Postgres). A public blockchain pool also behaves the same way, with Stellar being prioritized.

5.2 Discussion

5.2.1 Selection Process

Inspecting the results of the evaluation, scenarios one and two confirm what is to be expected. The framework chooses the cheapest implementation if the user prioritizes costs and it chooses the most performant implementation if the user prioritizes performance. The chosen blockchain implementations for this framework do have the special characteristic, that the cheapest one equals the most performant one. Though, this does not detract from the fact, that the framework does behave as expected in the first two scenarios. However, the behavior is different when the transaction split gets introduced into the policy. While it might make sense in case one wants the best performance, since transactions can be sent different blockchains simultaneously, which improves the performance compared to sending only to one blockchain, it does not make much sense from a cost saving aspect. Especially, since a complete split to all blockchains raises the cost exponentially, instead of minimizing it. To solve this issue, the split parameter would either have to be removed completely, if the user wishes to minimize costs or a second threshold needs to be introduced, where the user can configure to which blockchains the transactions should be split to. In addition, if the transaction costs exceed the threshold, the transaction would not choose said blockchain for the split.

5.2.2 Blockchain Characteristics

Another point of discussion are the different blockchain characteristics. Particularly, those that influence the performance. As mentioned in Section 3.2, different sources cite different TPS values for the blockchain implementations. For this prototype, generally the lower spectrum of reported TPS values have been chosen, to be on the safe side of performance values. A particular problem pose the private blockchains which can be configured to reach different TPS values. Thus, to use this framework, proper benchmarking would be needed

in advance, to get the exact values for the blockchain performances. The framework will use the blockchain characteristics provided in the database and if this data is flawed, the output is going to be flawed as well.

The costs also provide a point of discussion in this framework. As mentioned in Section 5.1.1, private blockchains and IOTA do not have any transaction fees, which causes the policies to never change, as soon as one of these implementations enter the blockchain selection pool (unless a time frame has been specified for the policy activity). For the evaluation, an arbitrary transaction cost of CHF 0.01 has been set, though for production use a thorough investigation about costs of these solutions should be performed. While the blockchain implementations do not charge any transaction fees, costs are still generated by usage of these implementations, be it costs for hosting or electricity costs, which are generated by participating in said networks.

As seen during the evaluation, the framework generally performs as expected when selecting the blockchains based on performance, costs and data confidentiality. Blockchains are versatile though and some of them offer specialized characteristics. While there is a parameter for filtering blockchains that are Turing complete, which allows to select blockchains that enable smart contracts, other specialties like degree of centralization are currently not taken into account during the selection. Though, there is correlation between throughput and centralization, it is still not choosable directly. With the rate the blockchains are appearing though, not all specific characteristics can be captured by the framework and in the future the focus might shift to different characteristics, which makes it a difficult task to focus on everything that is out there. Thankfully, the framework is expandable, which enables to support these characteristics in the future.

5.2.3 Stability and Security

During evaluation, out of the 4'000 requests made by the test script, three requests timed out, according to the default timeout of 2'000 ms set by Mocha. The reason for these timeouts was caused by one of the external resources for calculating transaction fees. For these three requests it took too long to get an answer from the external API. Since the framework is dependent on accurate cost data, it is essential that the data is as fresh as possible, as the value of blockchains changes at a rapid pace. Thus, on every request, a call is made to various APIs to make sure that the freshest data is fetched. The negative aspect is that the framework is dependent on those sources. When these are down, the framework is not able to get accurate data. Making the framework dependent on external resources always carries a certain risk, though in this case there is no way to circumvent this issue, unless the data is self-hosted, which would require getting hold of the real time data in some fashion.

Other than that, no issues arose within the framework. There was always only a single policy active, since policy conflicts are eliminated on creation, due to the validation taking place before the policy gets stored to the database. The requests always returned the data, its hash and the selected blockchain. Also, all sheets were properly processed and all transactions were stored in the database.

Security-wise, the framework currently does not support any authentication or authorization mechanisms. An implementation of such a mechanism depends on the usage scenario of the framework. While a classic registration process makes sense in case the framework is deployed in a public environment, in a private environment this is not an option as an authentication mechanism. The creation of users would have to be handled by the company operating the framework and then distributed to the customer. As for the authorization, JSON Web Tokens could provide help. JSON Web Tokens are a compact and self-contained way for securely transmitting information between parties as a JSON object [87]. Once the user is authenticated, a web token is received, which is encoded to contain the information about what the user is authorized to do. In the case of this framework, that would for example be the creation of transactions or the creation, modification or deletion of policies of a certain user. Though, before such a solution can be implemented, a clear concept should be created to determine who exactly should be able to create transactions or manage policies.

Chapter 6

Conclusion and Future Work

As presented in this thesis, selecting an appropriate blockchain implementation for a certain use case is an important, though not trivial task. To help achieve said task, this thesis introduces a Policy-based Blockchain Selection Framework, which recommends blockchain implementations, based on transaction information and pre-defined policies. These policies allow the user to define cost thresholds, performance and storage requirements towards the blockchains, access rights and other parameters to facilitate blockchain selection. These configurations set the rules the framework abides by to select blockchains.

The framework is divided into two core parts. A Policy Management Component that exposes a GUI on top of a REST API and a REST API endpoint that allows to pass transactions to the framework, called the Transaction Component. Users define policies via the Policy Management Component, which are then used to make a decision on a blockchain implementation, when a transaction is passed through the Transaction Component.

Evaluation of the prototype showed that the framework recommends blockchain implementations based on the focus the user sets towards the implementation. Be it blockchain type for secure data storage, cost minimization or performance optimization. Though, there are still some potential points of improvements and future work that are further discussed in Section 6.1.

6.1 Future Work

The prototype presented in this thesis provides a first step towards a Policy-based Blockchain Selection Framework. Still, there are some shortcomings and potential improvements, which should be taken care of before the proposed framework can be moved from a “prototype” status to a software used in production.

While the prototype allows the definition of multiple users, it does not support any authentication or authorization mechanisms currently. For production use, it needs to determine which parameters a certain user should be able to access and/or alter.

Furthermore, the prototype allows to split transactions into multiple blockchain implementations. For performance reasons this allows higher throughput and faster transmissions, though in turn the costs rise astronomically, since the transactions are distributed over all blockchain implementations, that abide the rest of the configured parameters. To enable more cost control, an additional parameter could be introduced, which enables to limit the amount of transactions to the most expensive implementations.

Another area that could be worked on is the currency management. Currently, the user is able to chose a single currency at the beginning, which is not switchable and is valid for all policies. Allowing multiple currencies, might be a feature that could be useful for certain use cases. For example in the case of cross-border payments, where there might be a need for calculation in multiple currencies.

Finally, usability-wise it can sometimes be frustrating to define policies, when no applicable blockchain implementation fits all the chosen parameters. For instance, when the user chooses that he wants to use a public blockchain but also chooses a minimum TPS threshold that is not supported by any of the public blockchains supported by the framework, the user gets an error message informing, that no blockchain implementations have been found that fit the chosen parameters. While this can be partially eliminated as soon as the framework supports additional blockchains, it could still be improved nonetheless. To solve this issue a bit more gracefully, the creation of the policies could be handled in a guided way, that walks the user throughout the creation of such a policy. Gradually removing the options that do not fit the previously chosen parameters.

Bibliography

- [1] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. <https://bitcoin.org/bitcoin.pdf>, 2008. Accessed: 2019-05-12.
- [2] M. Iansiti and K. R. Lakhani. *The Truth About Blockchain*. <https://hbr.org/2017/01/the-truth-about-blockchain>, January 2017. Accessed: 2019-05-12.
- [3] A. Sergeenkov. *How Blockchain is Changing Money Transfers*. <https://hackernoon.com/how-blockchain-is-changing-money-transfers-e9cb85e94932>, November 2018. Accessed: 2019-05-20.
- [4] Hochschule Luzern and Luxoft. *Auswertung der Blockchain-Konsultativabstimmung in der Stadt Zug*. http://www.stadtzug.ch/dl.php/de/5c00ff8dbb8a6/eVoting-Stadt_Zug_Abschlussbericht_DE.pdf, November 2018. Accessed: 2019-05-20.
- [5] EOS.IO. *EOS.IO Technical White Paper v2*. <https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md>, March 2018. Accessed: 2019-05-20.
- [6] V. Buterin. *Ethereum White Paper - A Next-Generation Smart Contract and Decentralized Application Platform*. http://blockchainlab.com/pdf/Ethereum_white_paper-a_next_generation_smart_contract_and_decentralized_application_platform-vitalik-buterin.pdf, 2014. Accessed: 2019-05-21.
- [7] M. Hussey. *Private Blockchains*. <https://decrypt.co/resources/private-blockchain>, 2019. Accessed: 2019-05-21.
- [8] B. Rodrigues, E. Scheid, and B. Stiller. Blockchains everywhere - a use-case of blockchains in the pharma supply-chain. *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, page 772–777, May 2017.
- [9] B. Rodrigues, E. Scheid, and B. Stiller. Toward a policy-based blockchain agnostic framework. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 609–613, April 2019.
- [10] T. Hegnauer. *Design and Development of a Blockchain Interoperability API*. Master’s thesis, Universität Zürich, February 2019.
- [11] V. Butarin. *Chain Interoperability*. <https://static1.squarespace.com/static/55f73743e4b051cfcc0b02cf/t/5886800ecd0f68de303349b1/1485209617040/Chain+Interoperability.pdf>, September 2016. Accessed: 2019-05-28.

- [12] N. Bauerle. *What is a Distributed Ledger?* <https://www.coindesk.com/information/what-is-a-distributed-ledger>, 2019. Accessed: 2019-05-28.
- [13] D. Puthal, N. Malik, S. Mohanty, E. Kougianos, and G. Das. Everything you wanted to know about the blockchain: Its promise, components, processes, and problems. *IEEE Consumer Electronics Magazine*, 7:6–14, July 2018.
- [14] R. Shaan. *Difference Between Traditional and Delegated Proof of Stake*. <https://hackernoon.com/the-difference-between-traditional-and-delegated-proof-of-stake-36a3e3f25f7d>, April 2018. Accessed: 2019-05-28.
- [15] T. Zilavy. *What Is A Hybrid Blockchain And Why You Need To Know About It?* <https://tinyurl.com/yyjdrwlo>, October 2018. Accessed: 2019-04-17.
- [16] D. Dob. *Permissioned vs Permissionless Blockchains: Understanding the Differences*. <https://blockonomi.com/permissioned-vs-permissionless-blockchains/>, July 2018. Accessed: 2019-04-17.
- [17] V. Buterin. *Public and Private Blockchains*. <https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/>, August 2015. Accessed: 2019-04-17.
- [18] L. Shiff. *Public vs Private Blockchains: What’s the Difference?* <https://www.bmc.com/blogs/public-vs-private-blockchain/>, October 2018. Accessed: 2019-04-17.
- [19] H. Perretta. *Private vs. Public and Permissioned vs. Permission-less*. <https://blocktonite.com/2017/06/27/private-vs-public-and-permissioned-vs-permission-less/>, June 2017. Accessed: 2019-04-17.
- [20] D. Massessi. *Public Vs Private Blockchain In A Nutshell*. <https://medium.com/coinmonks/public-vs-private-blockchain-in-a-nutshell-c9fe284fa39f>, December 2018. Accessed: 2019-04-17.
- [21] P. Tasca and C. J. Tessone. *Taxonomy of Blockchain Technologies. Principles of Identification and Classification*, May 2017.
- [22] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün Sirer, D. Song, and R. Wattenhofer. *On Scaling Decentralized Blockchains*. In *Financial Cryptography and Data Security*, volume 9604, pages 106–125, February 2016.
- [23] PCHAIN. *The History and Status of TPS in Blockchain*. <https://medium.com/pchain-org/dr-feng-cao-the-history-and-status-of-tps-in-blockchain-eb8a98a02cfb>, October 2018. Accessed: 2019-05-27.
- [24] Investopedia. *Block Time (Cryptocurrency)*. <https://tinyurl.com/y5uhyusq>, 2019. Accessed: 2019-05-27.

- [25] L. Marx. *Storing Data on the Blockchain: The Developers Guide*. <https://malcoded.com/posts/storing-data-blockchain/>, July 2018. Accessed: 2019-05-31.
- [26] Bitcoinfees. *Bitcoin Transaction Fees*. <https://bitcoinfees.info/>, 2019. Accessed: 2019-05-24.
- [27] Hesiod Services. *How do Bitcoin Mining Fees Work?* <https://www.bitcoinmining.com/bitcoin-mining-fees/>, 2018. Accessed: 2019-05-24.
- [28] EOS New York. *EOS Resource Planner*. <https://www.eosrp.io/>, 2019. Accessed: 2019-05-24.
- [29] Stellar Development Foundation. *Fees | Stellar Developers*. <https://www.stellar.org/developers/guides/concepts/fees.html>, 2019. Accessed: 2019-05-27.
- [30] BlockchainHub. *What is a Smart Contract? Auto enforceable Code - Blockchain*. <https://blockchainhub.net/smart-contracts/>, 2019. Accessed: 2019-05-28.
- [31] K. Christidis and M. Devetsikiotis. Blockchains and smart contracts for the internet of things. *IEEE Access*, 4:2292–2303, 2016.
- [32] BTC Inc. *Yes, Bitcoin Can Do Smart Contracts and Particl Demonstrates How*. <https://bitcoinmagazine.com/articles/yes-bitcoin-can-do-smart-contracts-and-particl-demonstrates-how/>, 2017. Accessed: 2019-05-28.
- [33] BTC Inc. *Script - Bitcoin Wiki*. <https://en.bitcoin.it/wiki/Script>, April 2019. Accessed: 2019-05-28.
- [34] Bitinfocharts. *Bitcoin Block Time chart*. <https://bitinfocharts.com/>, 2019. Accessed: 2019-04-12.
- [35] M. Bartoletti and L. Pompianu. An analysis of bitcoin op_return metadata. *Lecture Notes in Computer Science*, page 218–230, 2017.
- [36] Ethereum. *Ethereum Website*. <https://www.ethereum.org/>, 2019. Accessed: 2019-04-17.
- [37] StackExchange. *Is there a limit for transaction size?* <https://ethereum.stackexchange.com/questions/1106/is-there-a-limit-for-transaction-size>, 2018. Accessed: 2019-04-17.
- [38] Ethereum Wiki. *Ethereum Wiki*. <https://github.com/ethereum/wiki/wiki>, April 2019. Accessed: 2019-04-17.
- [39] ETHStats. *Ethereum Network Status*. <https://ethstats.net/>, 2019. Accessed: 2019-04-17.
- [40] Etherchain. *Etherchain - The Ethereum Blockchain Explorer*. <https://www.etherchain.org/>, 2019. Accessed: 2019-04-12.
- [41] Stellar Development Foundation. *Stellar - Develop the world's new financial system*. <https://www.stellar.org/>, 2019. Accessed: 2019-05-27.

- [42] K. McCollom. *How Many Transactions Per Second Can Stellar Process?* <https://www.lumenauts.com/blog/how-many-transactions-per-second-can-stellar-process>, January 2018. Accessed: 2019-05-27.
- [43] Stellar Development Foundation. *Stellar.org Dashboard*. <https://dashboard.stellar.org/>, 2019. Accessed: 2019-05-27.
- [44] Stellar Development Foundation. *Transactions | Stellar Developers*. <https://www.stellar.org/developers/guides/concepts/transactions.html>, 2019. Accessed: 2019-05-27.
- [45] Stellar Development Foundation. *Smart Contracts | Stellar Developers*. <https://www.stellar.org/developers/guides/walkthroughs/stellar-smart-contracts.html>, 2019. Accessed: 2019-05-27.
- [46] CryptoLions.io. *EOS - Network Monitor*. <https://eosnetworkmonitor.io/>, 2018. Accessed: 2019-05-27.
- [47] N. Graham. *Study Says EOS Maxes Out At 250 Transactions Per Second*. <https://www.ethnews.com/study-says-eos-maxes-out-at-250-transactions-per-second>, November 2018. Accessed: 2019-05-27.
- [48] EOS.IO. *What is the character limit of MEMO?* <https://github.com/EOSIO/eos/issues/4296>, 2018. Accessed: 2019-05-27.
- [49] G. Hertzog, E. Benartzi and G. Benartzi. *Bancor Protocol - Continuous Liquidity for Cryptographic Tokens through their SmartContracts*. https://storage.googleapis.com/website-bancor/2018/04/01ba8253-bancor_protocol_whitepaper_en.pdf, March 2018. Accessed: 2019-05-27.
- [50] Nat. *What is the Bancor algorithm and how it is used for RAM trading?* <https://eosio.stackexchange.com/questions/1317/what-is-the-bancor-algorithm-and-how-it-is-used-for-ram-trading>, 2017. Accessed: 2019-05-27.
- [51] IOTA Foundation. *The Next Generation of Distributed Ledger Technology | IOTA*. <https://www.iota.org/>, 2018. Accessed: 2019-05-27.
- [52] A. Gal. *The Tangle: an Illustrated Introduction*. <https://blog.iota.org/the-tangle-an-illustrated-introduction-4d5eae6fe8d4>, January 2018. Accessed: 2019-05-27.
- [53] DEXON Foundation. *Blockchain Comparison*. <https://github.com/dexon-foundation/wiki/wiki/Blockchain-Comparison#iota>, 2017. Accessed: 2019-05-27.
- [54] Phil. *What is the IOTA transaction data structure?* <https://iota.stackexchange.com/questions/100/what-is-the-iota-transaction-data-structure>, November 2017. Accessed: 2019-05-27.

- [55] Intel Corporation. *Introduction - Sawtooth Documentation*. <https://sawtooth.hyperledger.org/docs/core/releases/1.0/introduction.html>, 2017. Accessed: 2019-05-27.
- [56] Mason. *TokenSoft Announces Official Support for Hyperledger Sawtooth*. <https://medium.com/tokensoft/tokensoft-announces-official-support-for-hyperledger-sawtooth-ef5cc088c178>, October 2018. Accessed: 2019-05-27.
- [57] D. Middleton. *Hyperledger's Sawtooth Lake Aims at a Thousand Transactions per Second*. <https://www.altoros.com/blog/hyperledgers-sawtooth-lake-aims-at-a-thousand-transactions-per-second/>, March 2017. Accessed: 2019-05-27.
- [58] Intel Corporation. *IntegerKey Transaction Family - Sawtooth Documentation*. https://sawtooth.hyperledger.org/docs/core/releases/1.0/transaction_family_specifications/integerkey_transaction_family.html#state , 2017. Accessed: 2019-05-27.
- [59] G. Greenspan. *Multichain White Paper*. <https://www.multichain.com/download/MultiChain-White-Paper.pdf>, July 2015. Accessed: 2019-05-27.
- [60] Z. Hintzman. *Comparing Blockchain Implementation*. <https://www.nctatechnicalpapers.com/Paper/2017/2017-comparing-blockchain-implementations/download>, October 2017. Accessed: 2019-05-27.
- [61] MultiChain. *Smart contract showdown: Hyperledger Fabric vs MultiChain vs Ethereum vs Corda*. <https://www.multichain.com/blog/2018/12/smart-contract-showdown/>, December 2018. Accessed: 2019-05-27.
- [62] D. C. Verma. *Simplifying network administration using policy-based management*. *IEEE Network*, 16(2):20–26, March 2002.
- [63] E. C. Lupu and M. Sloman. Conflicts in policy-based distributed systems management. *IEEE Transactions on Software Engineering*, 25(6):852–869, November 1999.
- [64] M. Charalambides, P. Flegkas, G. Pavlou, A. K. Bandara, E. C. Lupu, A. Russo, N. Dulav, M. Sloman, and J. Rubio-Loyola. Policy conflict analysis for quality of service management. In *Sixth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'05)*, pages 99–108, June 2005.
- [65] M. Charalambides, P. Flegkas, G. Pavlou, J. Rubio-Loyola, A. Bandara, E. Lupu, A. Russo, M. Sloman, and N. Dulav. Dynamic policy analysis and conflict resolution for diffserv quality of service management. In *2006 IEEE/IFIP Network Operations and Management Symposium NOMS 2006*, pages 294–304, January 2006.
- [66] PostgreSQL. *PostgreSQL: The World's Most Advanced Open Source Relational Database* . <https://www.postgresql.org/>, 2019. Accessed: 2019-04-12.
- [67] W3C. *Relationship to the World Wide Web and REST Architectures*. <https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest>, February 2004. Accessed: 2019-05-20.

- [68] Docker Inc. *Docker: Enterprise Container Platform for High-Velocity Innovation*. <https://www.docker.com/>, 2019. Accessed: 2019-04-12.
- [69] CoinMarketCap. *Cryptocurrency Market Capitalization*. <https://coinmarketcap.com/>, 2019. Accessed: 2019-04-12.
- [70] EOS Canada. *EOS Canada: Leading Block Producer*. <https://www.eoscanada.com/>, 2019. Accessed: 2019-04-12.
- [71] Bitcoinfees. *Bitcoinfees: Predicting Bitcoin fees for transactions*. <https://bitcoinfees.earn.com/>, 2019. Accessed: 2019-04-12.
- [72] BlockCypher Inc. *Blockcypher: Blockchain Web Services*. <https://www.blockcypher.com/>, 2018. Accessed: 2019-04-12.
- [73] Node.js Foundation. *A JavaScript runtime built on Chrome's V8 JavaScript engine*. <https://nodejs.org/en/>, 2019. Accessed: 2019-05-20.
- [74] Google. *A game changer for interactive performance*. <https://v8.dev/>, 2019. Accessed: 2019-05-20.
- [75] L. Orsini. *What You Need To Know About Node.js*. <https://readwrite.com/2013/11/07/what-you-need-to-know-about-nodejs/>, November 2013. Accessed: 2019-05-20.
- [76] Y. Kadishay. *JavaScript V8 Engine Explained*. <https://hackernoon.com/javascript-v8-engine-explained-3f940148d4ef>, January 2019. Accessed: 2019-05-20.
- [77] Libuv team. *Cross-platform asynchronous I/O*. <http://libuv.org/>, 2019. Accessed: 2019-05-20.
- [78] D. Kahn. *What you should know to really understand the Node.js Event Loop*. <https://tinyurl.com/NodejsEventLoop>, July 2017. Accessed: 2019-05-20.
- [79] StrongLoop, IBM, and other expressjs.com contributors. *Express - Node.js web application framework*. <https://expressjs.com/>, 2017. Accessed: 2019-05-20.
- [80] Mozilla. *Nunjucks - A rich and powerful templating language for JavaScript*. <https://mozilla.github.io/nunjucks/>, 2019. Accessed: 2019-05-20.
- [81] J. Martin. *Managing the Data Base Environment*, page 381. A James Martin book. Pearson Education, Limited, 1983.
- [82] MongoDB Inc. *The most popular database for modern apps*. <https://www.mongodb.com/>, 2019. Accessed: 2019-05-20.
- [83] Xplenty. *The SQL vs NoSQL Difference: MySQL vs MongoDB*. <https://tinyurl.com/MySQLvsMongoDB>, September 2017. Accessed: 2019-05-20.
- [84] Mocha. *Mocha - the fun, simple, flexible JavaScript test framework*. <https://mochajs.org/>, 2019. Accessed: 2019-05-24.

- [85] Visionmedia. *SuperTest - HTTP assertions made easy via superagent*. <https://github.com/visionmedia/supertest>, 2019. Accessed: 2019-05-24.
- [86] Chart.js. *Chart.js - Simple yet flexible JavaScript charting for designers & developers*. <https://www.chartjs.org/>, 2019. Accessed: 2019-05-25.
- [87] Auth0. *JSON Web Tokens - Introduction to JSON Web Tokens*. <https://jwt.io/introduction/>, 2019. Accessed: 2019-05-25.
- [88] Techopedia. *What is an Application Programming Interface (API)?* <https://www.techopedia.com/definition/24407/application-programming-interface-api>, 2019. Accessed: 2019-05-29.
- [89] B. Fraser. *RFC 2196 – Site Security Handbook*. <https://tools.ietf.org/html/rfc2196#page-29>, September 1997. Accessed: 2019-05-20.
- [90] Binance Academy. *Byzantine Fault Tolerance Explained*. <https://www.binance.vision/blockchain/byzantine-fault-tolerance-explained>, December 2018. Accessed: 2019-06-03.
- [91] R. Fielding and J. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. <https://tools.ietf.org/html/rfc7231>, June 2014. Accessed: 2019-05-31.
- [92] Techopedia. *What does Interoperability mean?* <https://www.techopedia.com/definition/631/interoperability>, 2019. Accessed: 2019-05-31.
- [93] Techopedia. *What does Peer-To-Peer Network (P2P Network) mean?* <https://www.techopedia.com/definition/25777/peer-to-peer-network-p2p-network>, 2019. Accessed: 2019-05-31.
- [94] S. Zeil. *Turing Completeness*. <https://www.cs.odu.edu/~zeil/cs390/latest/Public/turing-complete/index.html>, 2016. Accessed: 2019-05-31.
- [95] Techopedia. *What does Web-Based Application mean?* <https://www.techopedia.com/definition/26002/web-based-application>, 2019. Accessed: 2019-05-31.
- [96] Techopedia. *What does Web Service mean?* <https://www.techopedia.com/definition/25301/web-service>, 2019. Accessed: 2019-05-31.
- [97] Docker Inc. *Docker docs*. <https://docs.docker.com/compose/install/#install-compose>, 2019. Accessed: 2019-04-12.
- [98] Nodemon. *nodemon reload, automatically*. <https://nodemon.io/>, 2019. Accessed: 2019-04-12.

Abbreviations

API	Application Programming Interface
BC	Blockchain
BTC	Bitcoin
CHF	Swiss Franc
CLI	Command Line Interface
CRUD	Create, Read, Update, Delete
dPoS	delegated Proof-of-Stake
ETH	Ethereum
EUR	Euro
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HYP	Hyperledger
I/O	Input / Output
JS	JavaScript
JSON	JavaScript Object Notation
MLC	Multichain
P2P	Peer-to-Peer
PBM	Policy-based Management
PoS	Proof-of-Stake
PoW	Proof-of-Work
PSG	PostgreSQL
REST	Representational State Transfer
SQL	Structured Query Language
TPS	Transactions Per Second
TTP	Trusted Third Party
USD	US Dollar
XLM	Stellar Lumens

Glossary

API An Application Programming Interface (API) is a set of protocols, routines, functions or commands used for software development or facilitation of interaction between distinct systems [88].

Authentication Authentication is the process used to identify a user towards an application [89].

Authorization Authorization is the process of granting privileges or access rights to processes and users [89].

Bancor Formula The Bancor Formula is a standard for Smart Tokens, ensuring continuous liquidity and real-time price discovery between blockchain-based assets [49].

Block Gas Limit The Block Gas Limit is the maximum amount of Gas all transactions inside a block are allowed to consume [6].

Byzantine Fault Tolerance Byzantine Fault Tolerance is the property of a system that is able to resist the failures derived from the Byzantine Generals' Problem [90].

Byzantine Generals' Problem The dilemma assumes that each general has its own army and that each group is situated in different locations around the city they intend to attack. The generals need to agree on either attacking or retreating. Communication is only possible through couriers, which means messages can get delayed, destroyed, lost or the courier could act maliciously [90].

Double Spending Attacks Double Spending Attacks refer to the attack in which a token is spent more than once [1].

Gas Gas is the internal unit for computation of Ethereum. It is used for executing transactions or deploying smart contracts [6].

HTTP The Hypertext Transfer Protocol (HTTP) is a stateless application-level protocol for distributed, collaborative, hypertext information systems [91].

Interoperability Interoperability is the property that allows for the unrestricted sharing of resources between different systems [92].

Peer-to-Peer A Peer-to-Peer network is a group of computers, each of which acts as a node for sharing data within the group. Instead of having a central server, each node serves the data [93].

RAM RAM is the internal unit of computation used for storing data in the EOS network [28].

REST Representational State Transfer (REST) is a architectural style that defines a set of constraints to be used for creating Web Services. It manipulates resources, using a uniform set of “stateless” operations [67].

Turing Completeness A computer programming language is said to be Turing complete if it can be used to simulate arbitrary Turing machines. It is used as a way to express the power of a data-manipulation rule set. The term is coined after the mathematician Alan Turing, the inventor of the Turing Machine. [94].

Web Application A Web Application is a program that is accessed over a network connection using HTTP, rather than existing within a device’s memory [95].

Web Service A Web Service is a software application with a standardized way of providing interoperability between disparate applications over HTTP [96].

List of Figures

2.1	Policy-based Blockchain-agnostic Framework architecture [9]	3
3.1	Policy-based Management [62]	13
4.1	Policy-based Blockchain Selection Framework architecture	18
4.2	Policy Management Component workflow	19
4.3	Main view of the Policy Management Component	20
4.4	Transaction Component workflow	21
5.1	Metrics of the evaluation of private blockchains	33
5.2	Metrics of the evaluation of public blockchains	34
5.3	Metrics of the evaluation of mixed blockchains including split transactions	35

List of Tables

3.1	Summary of blockchains and characteristics	12
4.1	Costs of blockchain transactions and calculation methods	25
4.2	Choosable parameters for a policy	28
5.1	Policies for the evaluation of private blockchains	32
5.2	Policies for the evaluation of public blockchains	33
5.3	Policies for the evaluation of mixed blockchains including split transactions	35

Appendix A

Installation Guidelines

A.1 Installation with Docker

The project can either be set up and developed with Docker or set up locally. Prerequisites for Docker are a working Docker installation [68] and Docker Compose [97].

To develop with Docker, skip to Section A.3 and execute one of the following commands, depending on what is needed:

```
1 #set up docker container
2 docker-compose up
3 #set up docker container and rebuild image
4 docker-compose up --build
5 #set up docker container and start in detached mode
6 docker-compose up -d
```

Listing A.1: Docker Compose commands

A.2 Local Installation (Alternative to Docker)

To start developing on a local installation, Node.js version > 10.x.x (LTS) [73] and MongoDB [82] need to be installed. The project also supports hot-reloading by using Nodemon. Nodemon is a utility that will monitor any changes in the source code and automatically restart the server [98]. For hot-reloading, Nodemon needs to be installed globally. The project includes a dump of the static blockchain data needed for this project in folder `/db-dump`. Executing the commands listed in Listing A.2 finalizes the installation process.

```
1 #install nodemon globally
2 npm install -g nodemon
3 #install all dependencies
4 npm install
5 #load blockchain data into the database
6 mongorestore --nsInclude policy-framework.*
```

Listing A.2: Setup commands for the framework

A.3 Configuration

Before this application can be used, two parameters have to be set as environment variables. One is used for authentication towards the CoinMarketCap API [69] and the other is the URL for the MongoDB database. In case the provided Docker configuration is used, the database URL corresponds to: `mongodb://mongo:27017/policy-framework`.

```
1 COINMARKETCAP_API_KEY= ""
2 DB_URL= ""
```

Listing A.3: Contents of the .env file

For local or docker usage, the provided `.env.example` file can be copied and renamed to `.env`. Then the corresponding values can be set. As long as the application is started in development mode, the application is going to search for values set in this file. For production usage the environment variables have to be in the environment.

A.4 Testing

The test script runs test cases, which create transactions via the REST API. To perform the tests, a running server is needed. Also pre-defined users and policies are necessary.

```
1 #run test cases
2 npm test
```

Listing A.4: Command for running tests

A.5 Run Server

```
1 #start application in debug mode with nodemon
2 #running on port: 3000, remote debug port: 3001
3 npm run devstart
4 #start application in production mode
5 npm start --production
```

Listing A.5: Command for running tests

A.6 Usage

The application consists of two different components. Entering `http://localhost:3000` into your browser leads to the GUI of the framework, which allows creation of user policies.

As soon as policies have been defined, data can be passed to the `/api/transactions` endpoint via HTTP POST request with the parameters presented in Listing A.6 or A.7. The resulting response consists of an array of objects. This array contains, among other data, the data to be stored in the blockchain and the corresponding blockchain it should be sent to.

```
1 Header :
2
3 "Content-Type": multipart/form-data
4
5 Body :
6 {
7     "username": String ,
8     "xlsxFile": File ,
9     "minTemp": Integer ,
10    "maxTemp": Integer
11 }
```

Listing A.6: Parameters for POST request with temperature file

```
1 Header :
2
3 "Content-Type": multipart/form-data
4
5 Body :
6 {
7     "username": String ,
8     "data": String
9 }
```

Listing A.7: Parameters for POST request with data as a string

Appendix B

Contents of the CD

The CD contains the following data and folders:

Code Contains the source code of the Policy-based Blockchain Selection Framework prototype including setup instructions. Equivalent to the GitHub repository found at <https://github.com/dlakic/bc-policy-selection>.

Data Contains the generated data of the evaluation part of this thesis. A database dump and statistics in JSON as well as HTML format.

Figures Contains figures and diagrams used in this thesis including their source files.

Presentation Contains the midterm presentation in Powerpoint (.pptx) format.

Thesis Contains the written part of the thesis as LaTeX source files as well as in PDF format.