FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Comparison of Ethereum and NEO as smart contract platforms

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Business Informatics

eingereicht von

Marco Bareis, BSc.
Matrikelnummer 00825264

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ass.Prof. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn Monika di Angelo

Wien, 5. November 2019

Marco Bareis | Monika di Angelo

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Comparison of Ethereum and NEO as smart contract platforms

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Business Informatics

by

Marco Bareis, BSc.
Registration Number 00825264

to the Faculty of Informatics

at the TU Wien

Advisor: Ass.Prof. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn Monika di Angelo

Vienna, 5th November, 2019

Marco Bareis Monika di Angelo

Erklärung zur Verfassung der Arbeit

Marco Bareis, BSc.

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

Wien, 5. November 2019

Marco Bareis

Kurzfassung

Aufgrund des Hypes um Bitcoin und Cryptocurrencys haben Blockchains in den letzten Jahren viel Aufmerksamkeit erhalten. Aber Cryptocurrencys sind bei weitem nicht die einzige Anwendung der Blockchaintechnologie. Smart Contract, also Applikationen die nach vordefinierten und unveränderbaren Regeln agieren, stellen eine weitere Anwendung dar. Solche Smart Contracts benötigen jedoch spezielle Platformen um ausgeführt werden zu können: so gennante Smart Contract Platforms. Die momentan meistverwendete Plattform is Ethereum, aber es gibt weitere Plattformen die interessante Alternativen darstellen. Eine vielversprechende dieser möglichen Alternativen is NEO. NEO ist in vielen Belangen ähnlich zu Ethereum, aber verspricht gleichzeitig einige Probleme zu lösen, mit denen sich Ethereum momentan konfroniert sieht – wie zum Beispiel die schlechte Skalierbarkeit.

Literatur, die sich mit den Unterschieden zwischen Ethereum and NEO befasst, ist spärlich. Vor allem NEO wird in der Literatur selten berücksichtigt, und falls NEO behandelt wird, dann passiert dies in der Regel nur oberflächlich. Außerdem haben die meisten Vergleiche und Evaluierungen von Smart Contract Plattformen beziehungsweise von Blockchains keine strukturierte Herangehensweise, sondern verwenden unterschiedliche Kriterien für unterschiedliche Plattformen. Das bedeutet, dass die meiste Literatur zu diesem Thema eine Übersicht der Plattformen darstellt, aber wenig Hilfe bei der Auswahl von Smart Contract Plattformen liefert. Diese Arbeit schließt diese Lücke, indem sie einen detalierten Vergleich von Ethereum und NEO durchführt. Um eine strukturierte Herangehensweise zu gewährleiten, wird in dieser Arbeit ein Kriterienkatalog basierend auf Kritierien in wissenschaftlicher Literatur abgeleitet. Dieser Kriterienkatalog wird anschließend auf die beiden Plattformen Ethereum und NEO angewandt um die für den Vergleich notwendigen Daten zu erhalten, die dazu dienen, die relevante Gemeinsamkeiten und Unterschiede zwischen Ethereum und NEO zu identifieren. Des weiteren ermöglicht dies eine Diskussion über die Auswirkungen dieser Unterschiede. Die Ergebnisse der Arbeit zeigen, dass obwohl Ethereum und NEO auf den ersten Blick sehr ähnlich zu sein scheinen, diese doch markante Unterschiede aufweisen. Die Unterschiede reichen vom allgemeinen Ziel der Plattform über die Reife der Dokumentation und Plattformfeatures bis hin zu praktischen Kritierien wie den Kosten für die Erstellung von Smart Contracts.

Abstract

Due to the hype around Bitcoin and cryptocurrencies, blockchains got a lot of attention in the last couple of years. However, cryptocurrencies are not the only application of blockchains. Smart contracts, which are applications that act on pre-defined rules, are another more generic use case of blockchains that gain popularity. Such smart contracts require smart contract platforms to be executed. Ethereum is currently the most popular smart contract platform, but there are other platforms that pose interesting alternatives. One of those promising platforms is NEO. NEO is similar to Ethereum in many ways and promises to solve issues that Ethereum currently faces like poor scalability.

However, literature about the differences and similarities between Ethereum and NEO is sparse. Especially NEO is rarely considered in scientific literature and if it is considered, its properties are only discussed superficially. Moreover, most evaluations and comparisons of smart contract platforms respectively blockchains do not use a structured approach, but rather apply different criteria to each platform. That means that most literature gives an overview of existing platforms, but does little to help developers to choose an adequate platform for their project. This thesis closes the gap by performing an in-depth comparison between Ethereum and NEO. In order to execute the comparison in a structured manner, this thesis derives a catalogue of criteria by extracting criteria used in scientific literature. This catalogue of criteria is then applied to Ethereum and NEO to obtain relevant information that is subsequently used to identify notable differences and similarities between Ethereum and NEO. Furthermore, it enables a discussion about the effects of those differences. The results show that although Ethereum and NEO seem to be quite similar, they differ in many key aspects ranging from the general goal of the platform over the maturity of its features and documentation to more practical aspects like the costs of creating smart contracts.

Contents

Kurzfassung **vii**

Abstract **ix**

Contents **xi**

1 Introduction **1**
1.1 Motivation 1
1.2 Problem Statement 2
1.3 Aim of the Work 2
1.4 Methodological Approach 3
1.5 Structure of the Work 3

2 Basics **5**
2.1 Blockchains 5
2.2 Smart Contracts 9
2.3 Decentralized Applications (DAPPs) 10
2.4 Overview of Ethereum 12
2.5 Overview of NEO 15

3 Catalogue of Criteria **19**
3.1 Collecting Criteria from Literature 19
3.2 Deriving the Catalogue of Criteria 21
3.3 Resulting Catalogue 22

4 Comparison of Ethereum and NEO **35**
4.1 Project 35
4.2 Blockchain Properties 42
4.3 Platform and Development 48
4.4 Execution and Operation 58

5 Discussion **67**
5.1 Key Differences 67
5.2 Applications and Markets 71

5.3 Sustainability 72

6 Conclusion **75**

A ERC-223 Token Source Code **77**

B NEP-5 Token Source Code **81**

List of Figures **85**

List of Tables **87**

Bibliography **89**

CHAPTER 1

Introduction

1.1 Motivation

In 2008 Satoshi Nakamoto published “Bitcoin: A Peer-to-Peer Electronic Cash System”, which laid the foundation of modern smart contract platforms by introducing the key concepts of the blockchain technology. “A blockchain is essentially a distributed database of records, or public ledger of all transactions or digital events that have been executed and shared among participating parties. Each transaction in the public ledger is verified by consensus of a majority of the participants in the system. [...] The blockchain contains a certain and verifiable record of every single transaction ever made”[30].

The first use cases built on blockchains were not smart contracts, but mostly related to cryptocurrencies. The most popular example is Bitcoin, which currently has a market capitalization of nearly 140 billion USD - far more than any other cryptocurrency [26]. Ethereum on the other side represents a type of blockchain which is not focused on cryptocurrencies, but on smart contracts and thus enabling financial and non-financial applications.

The term “smart contract” is described by the founder of Ethereum, Vitalik Buterin, as “systems which automatically move digital assets according to arbitrary pre-specified rules” [99]. Although the term “smart contract” first appeared in 1994 when Nick Szabo published his paper “Smart Contracts”, “it did not find usage until the notion of crypto currencies or programmable payments came into existence. Now the two programs, Blockchain and Smart Contracts can work together to trigger payments when a preprogrammed condition of a contractual agreement is triggered” [30]. Nowadays most smart contract applications on Ethereum can be classified as financial, notary or game contracts. Of course, most contracts cannot be classified at all, because the original source code was not published [8].

Since the publication of Ethereum, many other smart contract platforms were published. Some of them are specialized on a particular set of use cases - Ripple or Stellar for example are focused on being an electronic payment platform. Other platforms are limited to certain operation modes like private or permissioned chains.

1.2 Problem Statement

Although Ethereum is currently the most popular smart contract platform, other smart contract platforms propose an interesting alternative. One of the most promising platforms is NEO, which is often referred to as "Chinese Ethereum", because like Ethereum it allows a broad range of use cases but originates from China. Both platforms seem quite similar on the first glance, but a deeper analysis brings many differences between them to light like the consensus protocol, the language support or the overall philosophy and goal of the platform.

Since blockchain interoperability is still an issue, migrating distributed applications from one smart contract platform to another might be an expensive and time-consuming process. Therefore, selecting a smart contract platform should be an informed and long-term decision.

Although there is documentation about Ethereum and NEO available, there is no in-depth comparison of the functionality and mechanics of those two. This thesis aims to close the gap.

In order to provide a comparison of Ethereum and NEO, both platforms need to be evaluated and the following questions need to be answered:

- **RQ:** What are similarities and differences of Ethereum and NEO in respect to smart contract platforms?
  - **SQ:** What are relevant criteria to compare those platforms?

1.3 Aim of the Work

The aim of this thesis is to extend the current knowledge about the differences of Ethereum and NEO as smart contract platforms and discuss the effects of those difference on the development and operation of smart contracts or distributed applications so that an informed decision between those two platforms can be made.

To achieve this goal the following results will be produced:

- A list of criteria which can be used to compare Ethereum and NEO.
- An In-depth evaluation of Ethereum and NEO as smart contract platforms based on those criteria.
- A discussion about the effects of those differences on smart contract development.

1.4 Methodological Approach

Creation of a catalogue of criteria for comparison
First, a literature review about the Ethereum and NEO platforms themselves and the current limitations is required to gather the basic properties of these smart contract platforms. Then the existing literature about the evaluation and comparison of smart contract platforms and blockchains will be reviewed to gather the criteria for comparison.

Evaluation of Ethereum and NEO based on the derived catalogue of criteria
The catalogue of criteria will be applied to Ethereum and NEO. Further literature review will be required at this stage to gather more detailed information about the functionality and mechanics of those platforms.

Test concept and test runs using specifically created smart contracts
Some results might need verification using actual smart contracts on Ethereum and NEO. Therefore, based on the derived criteria smart contracts will be created and tested on the Ethereum test chains and the NEO test chain.

Out of scope / limitations:
The analysis is based on Ethereum and NEO as smart contract platforms. Cryptocurrency characteristics are only considered as far as they are relevant to the execution of smart contracts.

It is not the aim of the thesis to execute an in-depth performance test of NEO. The focus lies on the functionality of NEO and Ethereum.

1.5 Structure of the Work

The rest of this thesis is organized as follows:

Chapter 2 will introduce basic terms and mechanism of blockchains and smart contract platforms and will introduce both Ethereum and NEO. The following chapters will build on the information presented in this chapter.

Chapter 3 will deal with the creation of a catalogue of criteria which will be used to compare Ethereum and NEO. This chapter is based on a literature review of existing evaluations and comparisons in the area of blockchains. Furthermore, additional specific characteristics of NEO are considered as well, since NEO is usually not considered by existing comparisons.

In **chapter 4** the catalogue of criteria is applied to identify the differences and similarities of Ethereum and NEO. This chapter lays the foundation of the subsequent discussion of effects on smart contracts and distributed applications.

Chapter 5 presents the already mentioned discussion of the effects of the previous differences on the development and operation of smart contracts and distributed applications.

Finally, **chapter 6** contains the conclusion of the thesis and potential future work.

CHAPTER 2

Basics

Before we compare Ethereum and NEO, we need to define the basic terms and technology which this comparison is built upon.

2.1 Blockchains

“Blockchains are an emerging digital technology that combine cryptography, data management, networking, and incentive mechanisms to support the checking, execution, and recording of transactions between parties” [101].

The idea of the blockchain technology was first proposed by Nakamoto’s paper “Bitcoin: A Peer-to-Peer Electronic Cash System” in 2008. The goal of the paper was to create the basis for a payment system without any central financial institution. The key challenge was to prevent double spending attacks, which means that an attacker tries to spend more money or coins than they have available by sending two transactions in parallel to different recipients [108].

Blockchain technology supports creating a **distributed ledger** which can be replicated on nodes across the world. Those nodes are equal parts of the blockchain and operate the network together. There is no central entity controlling the blockchain, which means that there must be a mechanism in place to achieve consensus on the data between the nodes. The distribution of data also means that data loss because of the failure of a central database is not possible. Furthermore, public blockchains enable a high degree of transparency since all data is replicated on multiple nodes and publicly available [88].

Many types of transactions like payments or voting traditionally presume a trusted third-party to support the process. “Blockchains provide a different way to support these transactions. Instead of trusting third-parties, we would trust the collective jointly operating the blockchain and the correctness of their shared technology platform” [101].

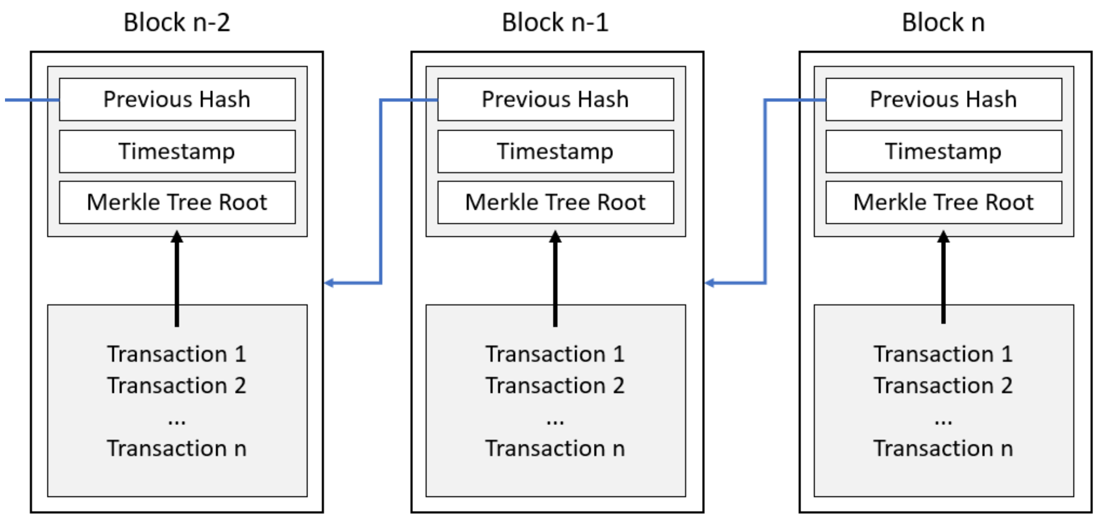

Figure 2.1: Blockchain data structure

The necessity of third-parties usually increases the costs of the transactions, which means that omitting those parties could reduce the costs of several processes.

The ledger or the state of a blockchain in general can only be changed by appending transactions. Transactions are added to the blockchain grouped together as **blocks**, which form a linked list. New transactions can be added using new blocks, but it is not possible to modify or delete existing blocks or transactions – a blockchain is therefore an append-only store of transactions [101]. To ensure this behavior each block contains a hash pointer to the previous block. A hash pointer contains the hash value of the linked block and acts as a pointer to that block. Therefore, blocks form a cryptographic chain. If the previous block would be modified, its hash value would change as well and would not fit the value of the hash pointer anymore – the link would not be valid anymore.

The exact structure of a block depends on the particular blockchain implementation, but there are certain elements that most chains share. The header usually contains – as already discussed – a hash pointer to the previous block, the timestamp of the block and the Merkle tree root. The Merkle tree root is the combination of all transaction hashes of that particular block and is used to verify that a transaction is actually part of a block as well as the order of the transaction. The body usually contains an ordered list of transactions, whose structure again depends on the particular implementation [101].

Since calculating a block requires a certain resource – like calculation power – it is not possible to recalculate a vast number of blocks. This concept also means that the more blocks come after a certain block the more secure it is. Together the hash pointer concept and the decentralized nature make a blockchain **immutable** [95]. Figure 2.1 visualizes a simplified version of the blockchain data structure.

Transactions usually contain the sender and recipient account (address) and either

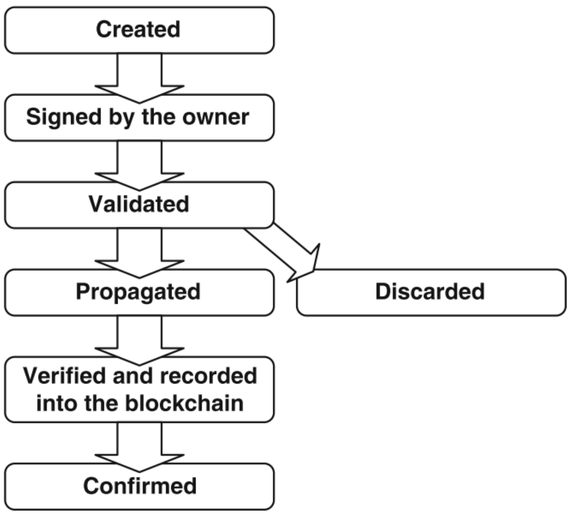

Figure 2.2: Transaction lifecycle [101]

the value to transfer or operation instructions. “Public key cryptography and digital signatures are normally used to identify accounts and to ensure integrity and authorization of transactions initiated on a blockchain. [. . .] Once created, the transaction is signed with the signature of the transaction’s initiator, which provides the authorization to spend the money, create a contract, or pass the data parameters associated with the transactions. A signed transaction should contain all the information needed to be executed” [101].

After the transaction was created and signed by the sender it has to be sent to a node of the blockchain network, which then verifies the transaction using the provided signature and the current value in the ledger. If it is valid, the transaction is propagated to the rest of the network. Eventually the transaction is included in a block. Depending on the blockchain, the block is either immediately confirmed or after a certain interval. Figure 2.2 summarizes the described generic transaction lifecycle.

The process of appending new blocks is called **mining** in most blockchain networks and is usually performed by a subset of the network nodes called “mining nodes”. Miners that append a new block will receive a reward or transaction fees – the exact process depends on the reward system and the consensus protocol of the network. Of course, multiple miners want to provide the next block, consequently a mechanism to decide on the next miner and therefore on the next block must be in place.

This mechanism is called **consensus protocol** and is part of the overarching network protocol. There are currently two widely spread mechanisms in place: Proof-of-Work and Proof-of-Stake [101]. Proof-of-Work requires the miner to solve a cryptographic puzzle before they can submit a new block to the network. The first miner solving the puzzle can add the next block to the blockchain. In this case the probability of a miner to be

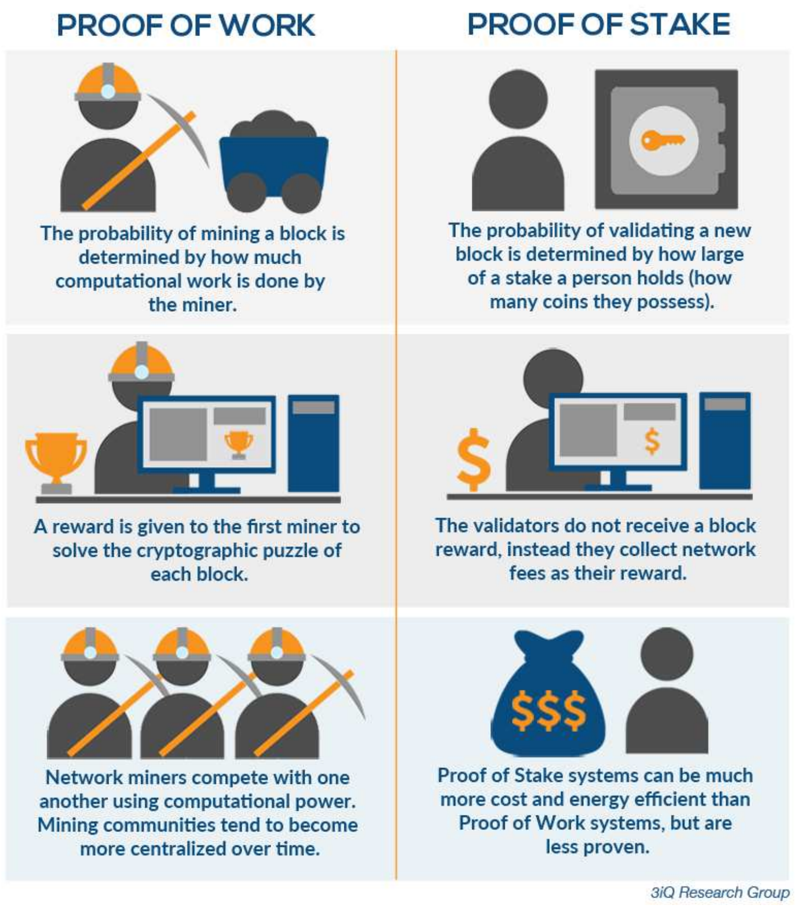
PROOF OF WORK
PROOF OF STAKE
The probability of mining a block is determined by how much computational work is done by the miner.
The probability of validating a new block is determined by how large of a stake a person holds (how many coins they possess).
A reward is given to the first miner to solve the cryptographic puzzle of each block.
The validators do not receive a block reward, instead they collect network fees as their reward.
Network miners compete with one another using computational power. Mining communities tend to become more centralized over time.
Proof of Stake systems can be much more cost and energy efficient than Proof of Work systems, but are less proven.
$$$
3iQ Research Group

Figure 2.3: Comparison of Proof-of-Work and Proof-of-Stake [1]

selected depends on their computation power. Since all miners are competing and trying to find the solution, this mechanism causes a high energy consumption in large networks. Proof-of-Stake on the other hand is not based on the computing power of a miner, but on their stake according to the blockchain's ledger. The higher their stake, the higher their probability to validate the next block. This mechanism normally depends on a certain degree of randomization to select the validator of the next block [108].

Since blocks are created in a distributed environment it is possible that two or more miners create a new block at the same time. In that case all mined blocks but one will be invalid eventually. Therefore, it is worth considering that even if a transaction is part of a block, it could be invalid in the future. Of course, this behavior depends on the particular blockchain and the consensus protocol used. Figure 2.3 shows the main differences between Proof-of-Work and Proof-of-Stake.

The **network protocol** which defines the consensus mechanism also "defines rights, responsibilities, and means of communication, verification, validation, and consensus across the nodes in the network. This includes ensuring authorization and authentication of new transactions, mechanisms for appending new blocks, incentive mechanisms (if needed), and similar aspects" [101].

The following list summarizes the **key characteristics** of blockchains:

- Blockchains provide a decentralization ledger whose cryptographic properties make it immutable.
- Changes on the ledger can be performed by adding new transaction. The sender of the transaction is verified using Public-Key-Cryptography.
- Transactions are grouped into blocks before they are added to the blockchain.
- New blocks are created by miners, who compete with each other. The selection of the miner depends on the consensus protocol.

2.2 Smart Contracts

As already mentioned in the introduction the first use cases built on blockchains were currency related with Bitcoin being the most successful example according to its market share [26]. But blockchains can support more functionality than just storing simple records and moving assets. Some blockchains allow software code to be stored and executed using blockchain transactions. Those programs are called smart contracts.

The term "**smart contract**" first appeared in 1994 when Nick Szabo published his paper "Smart Contracts", but the first application had to wait till the rise of the blockchains that served as platform for smart contracts [8]. Ethereum is currently the most used platform for smart contracts and the best-known example of smart contracts are tokens, like ICO or utility tokens.

The term smart contract is described by the founder of Ethereum, Vitalik Buterin, as "systems which automatically move digital assets according to arbitrary pre-specified rules" [99]. Which means that they allow the creation of trustless and automated agreements which can be considered fulfilled when pre-defined conditions are met. Those agreements rely on the features provided by blockchains [54]. They can hold and transfer digital assets, store data and invoke other smart contracts. The scope of the smart contract functionality depends on the blockchain's platform. Some blockchains like Bitcoin provide a very simple and limited set of instructions (e.g. no loops), which of course limits the functionality of smart contracts. On the other hand, Ethereum provides a Turing-complete environment, which in principle is expressive as any other typical programing language [101], [12].

The **creation of smart contracts** is done using blockchain transactions and its code cannot be modified after its creation – they are immutable. However, the state of the contract can be modified by calling methods that the smart contract provides. Although smart contracts are self-executing from a conceptional perspective, they cannot trigger any action on their own, but rely on external transactions.

Smart contracts already have many applications. They can be used to hold, administer and exchange digital assets on the blockchain or to implement customized cryptocurrencies. Further examples are gaming, voting, casinos or lotteries. The term “contract” implies that smart contracts are legal contracts, which is not the case, but they can be used to automate or monitor parts of legal contracts. As mentioned before the most used application of smart contracts are tokens. Tokens can represent real world assets like the share of a company or digital values like a currency in a video game [101].

Every transaction in a blockchain needs to be verified by all or at least some nodes, which means that smart contracts need to be executed on multiple nodes as well. Therefore, smart contracts need to have certain **basic properties**:

- Deterministic: A transaction that involves a smart contract needs to yield the same result every time it runs.
- Terminable: The execution of smart contracts needs to finish within a specified time limit. This is difficult to achieve in Turing-complete environments, because of loops.
- Isolated: Smart contracts can be uploaded by anyone who has access to the blockchain network. This means that smart contracts need to be isolated from each other, to avoid viruses or bugs spreading [54].

2.3 Decentralized Applications (DAPPs)

With the rise of platforms supporting smart contracts, the idea of decentralized applications (DAPPs) has gained popularity.

Most of the applications that we use today are built on a centralized model, which means that the decisions regarding of the application are made by a central entity – there is a central control. Some applications are distributed, which means that part of the application are exists in different locations [81]. There are various reasons for creating distributed applications: load balancing, scalability, resilience, increasing availability, etc. The opposite – non-distributed – therefore means that the application is hosted on one physical place.

A decentralized application is an application where decisions are not made by a single entity, but by multiple entities or nodes in a network. An application being decentralized does not automatically imply that the application is distributed. But decentralized applications which are based on smart contracts are both decentralized and distributed, since the underlying technology is a distributed ledger.

Table 2.1 summarizes the differences between decentralization and distribution and gives examples to illustrate those differences.

	Non-Distributed	**Distributed**
Centralized	Runs in one location and decisions are made by central entity. *Example: A proprietary word processing application for a local operating system. (e.g. Microsoft Word)*	Runs in multiple locations and decisions are made central. *Example: A cloud file storage provided by a single company. (e.g. Google Drive)*
Decentralized	Runs in one location and decision are made by multiple entities. *Example: An open source spreadsheet software for a local operating system. (e.g. LibreOffice Calc)*	Runs in multiple locations and decisions are made by multiple entities. *Example: A blockchain based cryptocurrency. (e.g. Bitcoin)*

Table 2.1: Distributed and decentralized applications

The platform stateofthedapps.com reports 60 new DAPPs published in July and 1.54 million transactions sent to DAPP contracts during the 24 hours of August 8, 2019. These numbers indicate the high interest in Distributed applications [93].

It is quite hard to determine the most common use cases for DAPPs, because checking the number and area of known DAPPs yields quite different numbers depending on the DAPP platform you use. The chart in figure 2.4 shows the number of known DAPPs according to the two well-known platforms stateofthedapps.com [93] and dapptotal.com [32].

It's clearly visible that the absolute numbers and the relation between the categories differ - according to dapptotal.com there are 1155 known DAPPs and stateofthedapps.com shows 2544 DAPPs for July 2019, which is a difference of approximately 120 %. This deviation could be explained by the fact, that it is difficult to categorize smart contracts and DAPPs, because not all of them have a published source code available [8]. Furthermore, the presented platforms use partly different categorizations, which had to be harmonized and this resulted in a larger "Other" category.

An example of one of the earliest DAPPs is Golem. Golem wants to enable resource sharing by utilizing idle computer power of network nodes. Bitcoin can also be seen as a DAPP, since it is a distributed application running on a blockchain – although the underlying blockchain was specifically designed to host that one application.

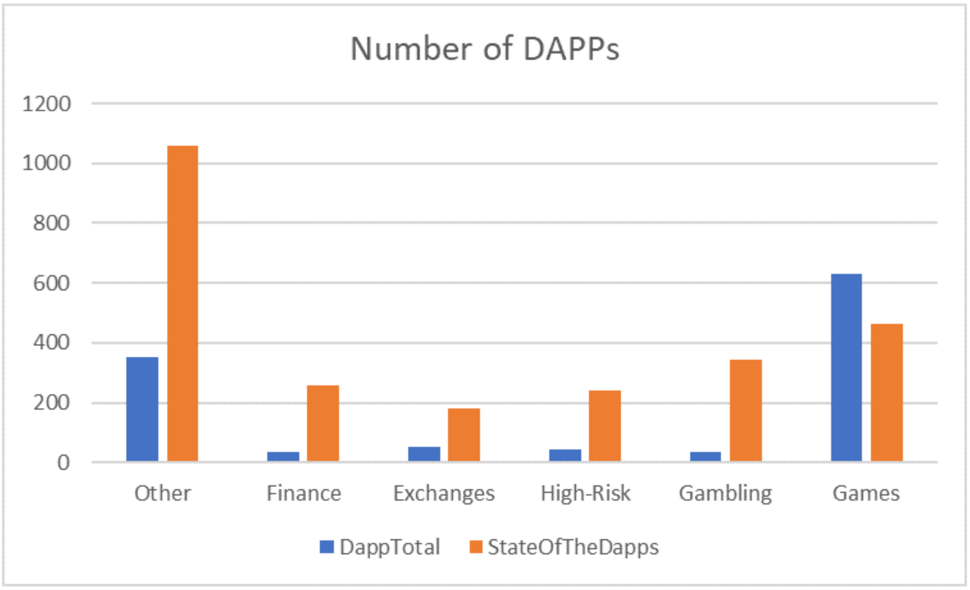

Figure 2.4: Number of DAPPs by Category

2.4 Overview of Ethereum

Bitcoin, the first application of blockchain, led to a variety of further blockchains aimed to provide financial transactions. Those blockchains are generally considered as the first generation of blockchains. After that, a new kind of blockchains emerged that provided a general-purpose platform with the support of programmable transactions and smart contracts – those blockchains are described as second generation blockchains. Ethereum is to the second generation blockchains what Bitcoin is to the first generation ones – the pioneer and current market leader [102].

According to the whitepaper written by Vitalik Buterin in 2013, Ethereum's intention is to be a general-purpose protocol for decentralized applications. To achieve this goal Ethereum built a blockchain with an integrated Turing-complete programing language which enables people to write powerful smart contracts including their own "arbitrary rules for ownership, transaction formats and state transition functions" [99].

Ethereum **philosophy** is described in five principles:

1. Simplicity: The Ethereum protocol should be as simple as possible, even at the cost of inefficiencies.

2. Universality: Ethereum does not provide features, but a versatile platform which can be used to develop DAPPs.

3. Modularity: The protocol should be built modularly, so that local changes do not affect the entire platform and features can be re-used.

4. Agility: Details of the protocol may change over time if new tests are conducted or new opportunities are found.

5. Non-discrimination and Non-censorship: The protocol should not try to restrict or prevent specific categories of smart contracts or DAPPs [99].

Those principals will play an important part in the comparison of Ethereum and NEO later.

Ethereum has a built-in cryptocurrency called Ether, which is used to fuel the network. Everybody who wants to create or use smart contracts or just create transactions in general has to pay transaction fees. Those fees are paid in Ether [85].

The state of Ethereum is stored in objects called **accounts**. Unlike bitcoin, the Ether balance is not stored as a list of unspent transaction outputs (UTXO) but as a property of those accounts [37]. Accounts are identified by a 20-byte and Ethereum distinguishes between externally owned accounts and contract accounts.

Externally owned accounts are controlled by people using private keys (see "Blockchains") and do not have code stored. Therefore, the only way to influence such an account is by creating and signing transactions. Contract accounts are controlled by code defined within the account and the behavior of the code can be triggered by sending messages to the contract's address [99].

Accounts contain the following fields:

- The nonce, which is basically a transaction counter and makes sure that each transaction is only processed once.
- The Ether balance.
- The contract code (if account is a smart contract)
- Storage, which is used by the contract code and empty by default [99].

A simple representation of a contract account can be seen in figure 2.5. It shows a contract account on the right side which is created and invoked by externally owned accounts (left side) using transactions.

Transactions in Ethereum refer to signed messages from externally owned accounts that are meant to change the state of the blockchain. The process of the state change is defined in the state transition function. The costs of a transaction depend on its complexity and is measured in "gas". The gas you pay is defined by the necessary calculation steps and the gas-price, which is given in Ether.

Transactions contain the usual fields to identify the sender and recipient and the Ether amount to transfer. Furthermore, transactions in Ethereum also contain a data-field,

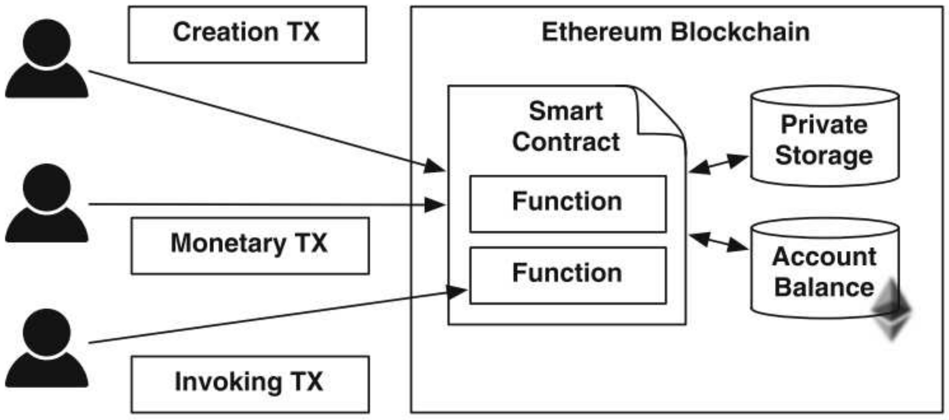

Figure 2.5: Smart Contracts in Ethereum [102]

which can be used to send data to a smart contract, a STARTGAS field which limits the calculation steps of a contract call and the GASPRICE field, which is the gas price the sender is willing to pay [100]. Both STARTGAS and GASPRICE are used to make sure that one call cannot hog all resources and run indefinitely – it makes contracts terminable.

When a smart contract is created or called, its code is executed in the Ethereum Virtual Machine (EVM). The EVM executes the EVM code, which is a low-level, stack-based language. The EVM makes sure that the contracts are isolated from each other. Furthermore, the contracts code can only access a limited scope of resources, so that contracts maintain determinism [99]. Smart contracts for Ethereum are usually written in a higher java script like language called Solidity, which is currently the primary programming language.

Ethereum currently uses a Proof-of-Work based **consensus protocol**. The algorithms used by Ethereum is called "Ethash" and its proof is a cryptographically secure nonce that proves that the miner conducted a certain amount of calculation. The hashed value has to be below a certain threshold, which is described as difficulty, to be valid. The difficulty influences the block time since a smaller threshold means a higher calculation effort and therefore more time is needed provided the computational power stays on the same level. The difficulty is usually adjusted after every block so that the block time stays roughly at the same level [100]. The current block time in the Ethereum network is about 13 seconds [44].

Since the block time is rather short compared to other blockchains, stale blocks happen regularly. Stale blocks happen when two or more nodes find a valid nonce at roughly the same time and the solution of one node could not be propagated to the other successful nodes before they found their solution. All but one block have to be dismissed eventually – those blocks are called stale. To define which block is allowed to remain, Ethereum uses a simplified protocol based on GHOST. This means that the "heaviest" chain, the one with the most calculation done, can remain [100].

The next version of Ethereum called "Serenity" or Ethereum 2.0 will change the networks consensus mechanism. The new algorithm will be based on Proof-of-Stake, rather than Proof-of-Work. This means that the next block is not determined by the use of computational resources, but by voting. The reasons for the change are the high power consumption, the risk of centralization and the fear of 51% attacks. The weight of one's vote depends on the amount of Ether that is held by the account [45].

2.5 Overview of NEO

Ethereum is currently the most popular smart contract platform [93], but there are several known issues and limitations, like the scalability, that the platform currently faces. NEO is compared to Ethereum relatively unknown, but proposes an interesting alternative, because it aims to solve issues Ethereum currently has.

The **history** of NEO starts in China in 2014 when it was first founded under the name "Antshares". Three years later it was rebranded to "NEO" as a part of a relaunch campaign. Similar to Ethereum it has its own cryptocurrency – actually cryptocurrencies – and provides a Turing-complete smart contract platform [89].

The similarities with Ethereum stop when you look at the **overall philosophy** and goal of the platform. As mentioned before, Ethereum aims to be a fully decentralized featureless platform. NEO's goal on the other hand is to build a smart economy, which does not directly restrict developers when creating arbitrary smart contracts, but gives the platform a clear focus. The approach is described by NEO as "[...] the use of blockchain technology and digital identity to digitize assets, the use of smart contracts for digital assets to be self-managed, to achieve 'smart economy' with a distributed network." [70] or in a short version as "Digital Assets + Digital Identity + Smart Contract = Smart Economy" [70].

Digital assets related to NEO are programable assets in digital form that are stored in a decentralized manner leveraging the properties of blockchains (see chapter 2.1). NEO distinguishes between global assets, which can be identified by all smart contracts and are stored using the UTXO model, and contract assets, which are part of a smart contract and need a compatible client to be recognized. Contract assets are stored in an account-based model. Those assets can be linked to physical assets using the digital identity feature of NEO. The digital identity feature is based on the X.509 standard [70]. Of course, this does not only allow to link digital assets but also accounts to legal entities. Another key aspect of NEOs smart economy are smart contracts, which are the basis for transactions and agreements among parties.

Furthermore, NEO focuses on being regulatory compliant which again is leveraged by the ability to integrate digital identities. This is in strong contrast to most other blockchains

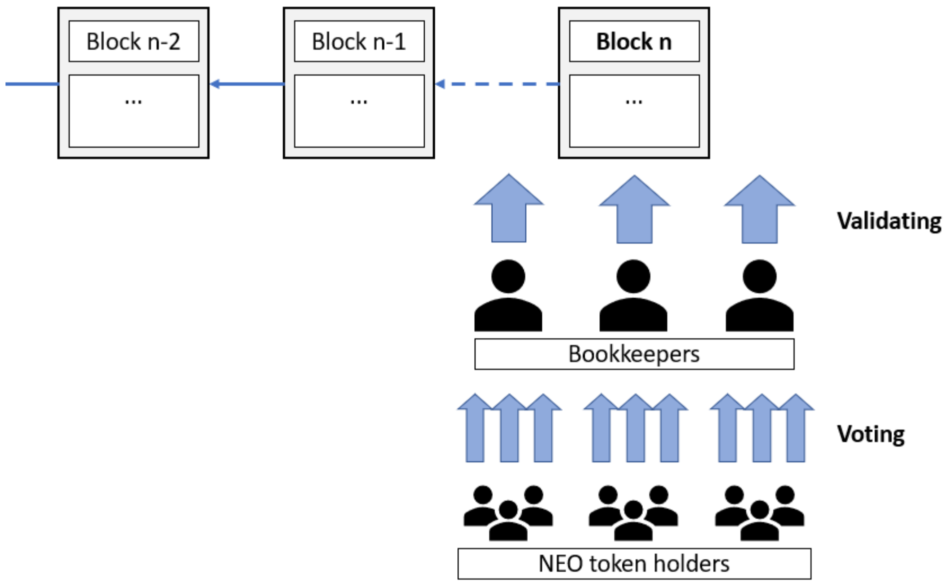

Figure 2.6: dBFT proxy voting

which want to achieve a fully distributed system without the interference of third-parties like governments.

NEO as a platform provides two kinds of **native tokens** or cryptocurrencies. The first token called "NEO" has a total supply of 100 million and represents the ownership of the NEO blockchain respectively the right to manage it. It can be used for governance, to create blocks and vote for bookkeepers, which are relevant in the consensus protocol. The second token is called "NeoGas" (GAS) which is used by users to fuel their transactions - like Ether in the Ethereum network. Since NEO provides a Turing-complete environment, it is also a part of the resource control mechanism of the platform. NeoGas is paid to accounts holding NEO tokens similar to dividends in the equity market [98], [70].

NEO uses the Delegated Byzantine Fault Tolerant (dBFT) **consensus mechanism**. This kind of protocol is based on proxy voting. As mentioned before, the holders of the NEO token are allowed to manage the network and vote for bookkeepers. This voting process is a continuous process. The selected bookkeepers are tasked to reach consensus on the new state of the blockchain together and therefore to create new blocks. The process is illustrated in Figure 2.6. This process requires that the bookkeepers' identity is known and verified by NEO – no matter if it is an individual or an institution [70]. "Thus, it is possible to freeze, revoke, inherit, retrieve, and ownership transfer due to judicial decisions on them. This facilitates the registration of compliant financial assets in the NEO network. The NEO network plans to support such operations when necessary" [70]. This highlights the strong direction of NEO towards compliance and the integration of third-parties like governments.

dBFT provides a high degree of finality which means that final blocks cannot become stale or forked. Therefore, a low block confirmation time can be achieved, which has some positive aspects for real-world application as we will see in the later chapters.

CHAPTER 3

Catalogue of Criteria

To enable a discussion on the differences of Ethereum and NEO, a structured comparison of both platforms is needed. Therefore, the first intermediate goal is to create a catalogue of criteria. In this chapter I will derive the criteria, which will be used for comparison later, by reviewing existing literature dealing with the comparison of blockchains respectively smart contract platforms.

3.1 Collecting Criteria from Literature

Since each source used at least partly different terms for the same criteria, I summarized those terms under the most common term, which should not affect the outcome of this thesis in any way. For example, the criteria describing if a platform supports public, private or a consortium chains is sometimes called “operation modes”, “blockchain types” or “deployment types”. In that example those different terms were summarized as “deployment types”.

Many papers did not have a structured list of criteria but looked at different platforms individually and listed the most important facts for each platform, which means that different criteria were applied to each platform. However, the used criteria on the individual level were considered as well if they could be applied to smart contract platforms in general.

The review resulted in 30 different criteria, which were used to compare blockchains respectively smart contract platforms. The list of criteria including their frequency can be seen in Table 3.1.

The results show that there are six criteria that are present in approximately half or more of all relevant papers: Consensus protocol, language support, block time, transaction fees, deployment types and throughput. The consensus protocol is the most discussed feature of smart contract platforms and blockchains in general, because it influences scalability

Criterion	**Frequency (rel.)**	**Frequency (abs.)**
Consensus protocol	73%	11
Language support	53%	8
Block time	53%	8
Transaction fees	53%	8
Deployment types	47%	7
Throughput	47%	7
Platform objectives & strategy	40%	6
Project status	33%	5
Platform interoperability	33%	5
Block confirmation time	33%	5
Turing completeness	27%	4
Smart contract support	27%	4
Cryptocurrency performance figures	27%	4
Storage architecture	27%	4
Reward system	27%	4
Public chain available	20%	3
Blockchain dependencies	20%	3
Execution Model	20%	3
Account structure	20%	3
Communication process	20%	3
Domain focus	20%	3
Origin and organization	13%	2
Developer community	13%	2
Market capitalization	13%	2
Identity management	13%	2
Platform governance	13%	2
Size of chain data	7%	1
ICO market share	7%	1
Forking support	7%	1
Tool support (Development)	7%	1

Table 3.1: Criteria collected from literature

and performance. Likewise, most of the other frequent criteria had a direct influence on the scalability of platforms as well.

3.2 Deriving the Catalogue of Criteria

After collecting the used terms from literature, the next step is to remove the criteria which do not influence the development, execution or distribution of smart contracts or DAPPs.

First, the criteria “size of chain data” was removed, because the amount of data stored by a node might be relevant for a node operator or miner, but not for the development of smart contracts. Because of similar reasons the “storage architecture”, “communication between nodes” and the “reward system” criteria were removed as well.

The “smart contract support” per se does not need to be investigated, since Ethereum and NEO were selected because they are smart contract platforms. Although the “account structure” criterion could be relevant when evaluating smart contract platforms, it was removed anyway, because its relevant properties for smart contract platforms are already included in other criteria.

Furthermore, the “cryptocurrency performance” and “market capitalization” are excluded because they relate more to cryptocurrencies than to smart contracts. The current “ICO market share” won’t be investigated as well, since it is just a snapshot and does not show the potential capabilities of smart contract platforms.

The rest of the criteria is considered worth investigating.

As already mentioned in the introduction most literature regarding the comparison of smart contract platforms considers Ethereum but not NEO. Therefore, unique attributes of NEO, which might be worth comparing to Ethereum could be missing. Consequently, further sources that analyze NEO as a smart contract platform were used to extend the already existing criteria.

According to the whitepaper NEO wants to leverage digital assets and digital identities to achieve a “smart economy” using a distributed network [70]. Accordingly, the integration of both digital assets and digital identities in the platforms are relevant criteria for this thesis.

Furthermore, the execution model of both chains needs to be investigated. This criterion is used to analyze how smart contracts are executed and how the basic characteristics like determinism and terminability are ensured.

NEO uses a different set of application standards than Ethereum. This thesis will include a comparison of the scope of Ethereum’s “Ethereum Request for Comments” (ERC) and NEO’s "NEO Enhancement Protocol” (NEP) [60].

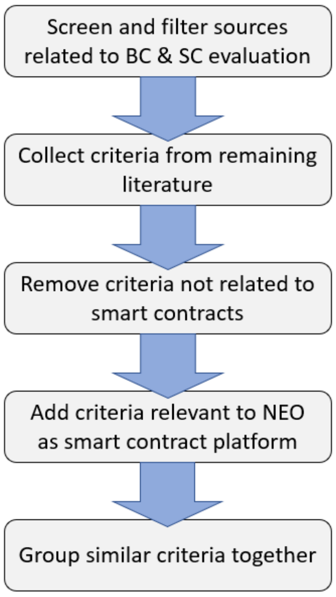
Screen and filter sources related to BC & SC evaluation
Collect criteria from remaining literature
Remove criteria not related to smart contracts
Add criteria relevant to NEO as smart contract platform
Group similar criteria together

Figure 3.1: Catalogue creation process

Other important aspects like the unique consensus protocol, its integration of cross platform interoperability or the origin of the organization behind NEO is already covered by the collected criteria from the chapter 3.3.1.

As a final step the criteria were grouped together. The overall process is summarized in Figure 3.1:

3.3 Resulting Catalogue

The catalogue of criteria, which was produced by the steps mentioned before, is now discussed in detail. The gathered criteria were grouped together so that an efficient comparison and discussion is possible.

3.3.1 Project

This criterion discusses properties of the overarching project or company that signs responsible for governing and developing the concerned platform. Criteria like the project maturity or the general orientation of the project have a direct influence on the selection process of smart contract platforms and are therefore relevant for this comparison.

Objectives & Strategy

This criterion discusses the general objectives of the platform's surrounding project and therefore on which specific use cases or application domains it focuses. Furthermore, it discusses potential goals the platforms want to achieve.

It is a crucial step to ensure that the smart contract platform's general orientation fits your application or smart contract. For example, if you plan on building a decentralized game, you should not use a smart contract platform that focuses on crowdsourcing and tokenized trading like the platform Waves [55]. Even if the platform supports your needs in the present, it might not do so anymore in the future. Assessing the fit of the platform's objectives to your application reduces this risk.

Another aspect which should be considered is the platform's strategy regarding anonymity and decentralization. Different applications may require different degrees of anonymity and it should be considered as well that future users may demand an anonym environment or exactly the opposite. "For example, while the majority of active Bitcoin users report minor concerns with the network's anonymity, almost 20 % consider abandoning the technology because of it" [83].

Connected to the topic of anonymity is the question of the degree of decentralization. Since smart contract platforms are usually decentralized, this question refers rather to the possibility of integrating third-parties or government bodies. The integration of third-party system may help to implement support services such as know-your-customer or money-laundering prevention [83]. If a DAPP requires such services or other third-party systems, it is worth considering the platforms strategy regarding decentralization.

Project Status

All major smart contract platforms and blockchains are being continuously improved over time and pass different stages which extend the platform's feature set. Usually blockchains first provide a test environment – also called testnet – before they launch their live system or mainnet, which is suitable for live applications [55].

Therefore it is valid to discuss whether the smart contract platform's mainnet is already launched and if the current stage of the platform can be considered mature enough to host production applications [8]. To answer those questions, we need to evaluate the overall process and the current stage of the platform.

As a part of the evaluation of the platform's maturity it is also worth considering the status of the documentation and specification, because more detailed information about the platform and its mechanics might be required by developers when creating DAPPs.

Origin and Organization

The origin country of a blockchain could be a relevant factor when selecting a platform for smart contracts. A prominent example which is currently discussed are security token. But evaluating the legal implications is difficult because "legislative bodies in most countries have not yet issued corresponding regulations if they are in fact ever introduced" [55].

Another example is China's relationship to cryptocurrencies which is often described as confusing and tumultuous. After banning crypto exchanges in 2017 and denying them the status of currency in general, the Hangzhou court classified Bitcoin as virtual property [10]. Although this does not influence the actual creation of smart contracts or DAPPs it could influence the reach or distribution capabilities of the application. If for example China is the main market for the application, one should consider using a platform that has a proven relationship to the Chinese government body to reduce the risk of future bans and other regulatory obstacles.

Governance

As mentioned before, blockchains or smart contract platforms adapt over time. A crucial question behind this ongoing transformation is who can propose changes and who finally decides which changes are being implemented.

"[...] The topic of blockchain governance has become central in the blockchain community, notably in the aftermath of the attack on 'The DAO'—a decentralized venture capital fund that was meant to be the world's first fully functioning decentralized autonomous organization" [82].

Blockchain governance can be achieved by a variety of rules which are usually classified as "on-chain" or "off-chain" governance. Most platforms incorporate both systems on different levels.

"On-chain governance refers to rules and decision-making processes that have been encoded directly into the underlying infrastructure of a blockchain-based system. This type of governance defines the rules of interactions between participants through the infrastructure within which these interactions take place; these interactions are solely determined by rules embedded within the underlying blockchain code—the so-called rule of code" [82]. Such a process might be implemented using coin-based voting.

"Off-chain governance comprises all other (i.e. non-on-chain) rules and decision-making processes that might affect the operations and the future development of blockchain-based systems. Off-chain governance includes both endogenous and exogenous rules. The former category refers to the rules adopted by a reference community to ensure the proper functioning and ongoing development of a blockchain-based system (including procedures to implement protocol changes). The latter category includes all rules imposed by a third-party onto the reference community, e.g. national laws and regulations, contractual agreements, technology standards, and so forth" [82].

The different governance methods are summarized in Figure 3.2.

The impact of the governance rules of a smart contract platform depend not only the rule set itself, but also the development roadmap of the project and its current status as well as on the strategic orientation of the platform.

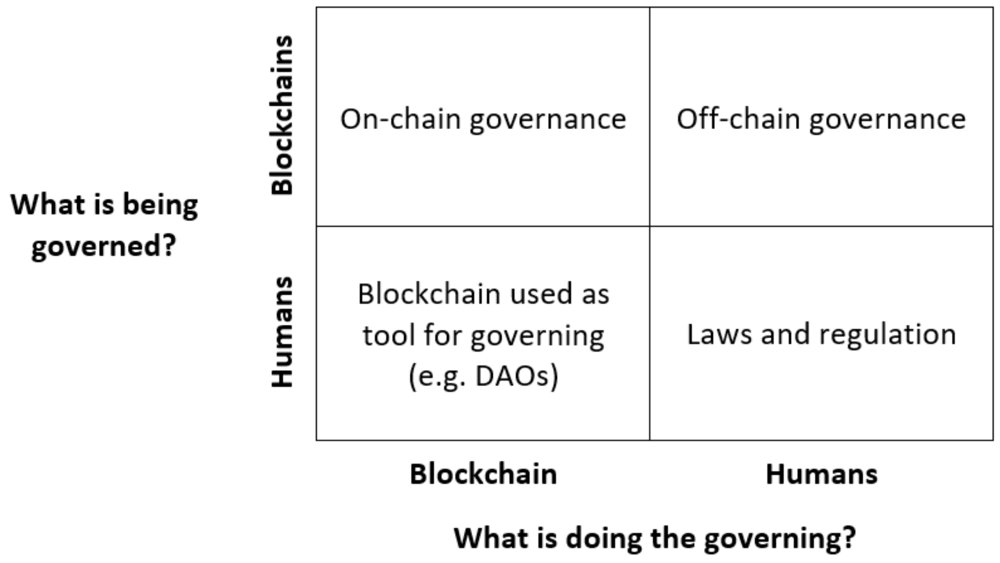

Figure 3.2: Governance Overview

3.3.2 Blockchain Properties

Besides the overarching project it is also necessary to discuss the key-technology behind most smart contract platforms: the blockchain. The parameters and mechanics of a blockchain are the most influential factors on smart contract development and execution, because amongst other things they limit the scalability of applications and define the cost structure.

Public Chain and Dependencies

One of the key criteria to select a smart contract platform is whether there is a public chain in operation. To qualify as public chain "it must be possible to write a contract and test it, or to explore the blockchain through some tools, or to run a node."[8] Thus enabling the chain to be accessed without special permission by anyone following the respective protocol. Such chains are also called permissionless [12].

Although most popular platforms operate a public chain, some platforms require you to create a network on your own. This factor should be considered when selecting a smart contract platform. If your application is based on a limited number of participants, a public chain might not be needed, but if your application targets a broad public audience, using a public chain might be the only choice in terms of distribution.

Furthermore, it is important to consider "whether the platform has its own blockchain, or if it just piggy-backs on an already existing one" [8]. If a smart contract platform relies on external blockchains, the limitations of those blockchains need to be considered as well.

Supported Deployment Types

Similar to the availability of a public chain, the supported deployment types or operation modes are another important factor which limits the potential applications of the smart contract platform. The common classification of deployment types divides networks into public, private and permissioned.

The most used deployment type of networks is “public”, which enables blockchains to be accessed by anyone on the internet. Most cryptocurrencies use public blockchains [103].

Private networks are controlled by a single organization or a group of organizations and the main difference to a public chain is the controlled access [12]. Therefore, it is not possible for everyone to join the network, read the contents of the ledger and add new transactions. In a private blockchain network, write permissions are often kept within one organization. The centralization allows a higher degree of flexibility [103].

Another operation mode is called permissioned or consortium mode, which describes blockchains that incorporate patterns of the public and private deployment types. Therefore, it is possible for everybody to read the chain, but only a few nodes are actually allowed to send and verify transactions [103].

Although public networks are usually in the focus of attention, to deploy networks in a private or permissioned mode can be crucial for many applications where writing or reading data is limited to certain parties or a higher degree of flexibility is needed. In the case of cryptocurrencies the desired properties are large-scale decentralization and security, which will usually achieved at the cost of scalability [59]. But in private or consortium chains, where the access is (partly) permissioned and the behavior of the participants can be controlled by off-chain means like legal contracts, full decentralization is usually not the most favorable property [59]. This allows more ways to optimize performance.

Table 3.2 summarizes the key differences of the mentioned deployment types.

Consensus Protocol

The criterion to evaluate smart contract platforms and blockchains mentioned most in general is the consensus protocol. This is due to the fact that the consensus mechanism is amongst other things the main driver of the transaction volume, energy consumption and operation costs.

The consensus protocol is a crucial part in the updating process of blockchains and ensures the reliability in a network of unreliable nodes. It guarantees the integrity and consistency of the blockchain across geographically distributed nodes and ensures that the next block in a blockchain is the one and only version of the truth [104], [7].

	Public	**Private**	**Permissioned (or Consortium)**
Configured for	Everyone	Specific organization	Consortium where data can be public or private
Managed by	Anyone	Organization itself	Multiple organizations
Centralized	No	Yes	Partially
Flexibility	Low	High	Moderate
Example	Bitcoin	Internal blockchain for accounting department	Hyperledger

Table 3.2: Deployment types

Bitcoin, the first popular blockchain, introduced Proof-of-Work (PoW) as part of the consensus mechanism and thus made it as the de facto-standard for most chains to follow. But Proof-of-Work has some major drawbacks. It is based on solving a cryptographic puzzle, which causes high costs and energy consumption. Therefore naturally other consensus mechanism were proposed and implemented [7].

There are other criteria directly linked to the specific consensus protocol besides the energy consumption that currently costs billions of dollars per year. For example, one factor is the fee structure, which should make operating a node profitable, so that a high number of nodes maintain the ledger [104].

Another frequently mentioned aspect is the transaction volume. The current limit of the transaction volume in many blockchains is far below classic not-decentralized platforms. For example, Ethereum currently processes up to 15 transactions a second. On the other hand Visa's network processes up to 24.000 transactions a second [104].

Furthermore, the consensus protocol has direct influence on the verification speed, which describes the time needed to validate a new transaction [4].

Nowadays "the most common consensus algorithms include Proof of Work (PoW), Proof of Stake (PoS), Delegated Proof of Stake (DPoS), Ripple, Practical Byzantine Fault Tolerance (PBFT) and Delegated Byzantine Fault Tolerance (dBFT)" [104].

When considering Ethereum and NEO the following consensus mechanisms are relevant:

Proof-of-Work (PoW)
"In a proof-of-work mining system, participants solve an asymmetric computation that is hard to decipher, but easy to check. One criticism of this algorithm is the overall cost of the processing power used to solve the problem. The solution is checked by other nodes and encoded onto the ledger, with a fee awarded to the miner." [55].

Although Proof-of-Work's performance is quite low and it is usually not considered as suitable for very large ecosystems, it is one of the first utilized consensus protocols and still widely used by several blockchains. Furthermore, it is exposed to 51 % attacks, which means it is possible to manipulate the ledger when 51 % of the computing power is available to the attacker. However, this kind of attack would require an immense amount of money in most blockchains [104], [4].

Proof-of-Stake (PoS)
To reduce the high costs of the consensus process, Proof-of-Stake was introduced. "PoS defines a consensus mechanism, where owners of a crypto currency have to prove ownership (proof for their stake). [For example] a user with 1% of the crypto currency can create [or propose] 1% of the blocks" [12]. This consensus protocol works without the need to solve a cryptographic puzzle and thus consumes less power than the Proof-Of-Work mechanism.

The details of this mechanism vary depending on the particular blockchain implementation. Usually the validators must own the required amount of coins for a specific duration of time before they can participate in the validation process. Furthermore, the selection of the proposer of a new block depends on the individual blockchain. In some chains the proposer is assigned by a round-robin approach [4], [55].

Delegated Byzantine Fault Tolerance (dBFT)
In the Delegated Byzantine Fault Tolerance algorithm only a subset of nodes is responsible for the execution of the consensus protocol. To achieve credibility and trust, those nodes must deposit a certain amount of money as collateral. Blockchains using this algorithm have to find the right size of collateral. If the amount is too low, an attack is possible. If the amount is too high, the liquidity of the entire blockchain could suffer [55].

This consensus protocol is "characterized by high throughput, high scalability and low costs, which make them suitable for infrastructure blockchain" [55].

3.3.3 Platform and Development

This section collects the criteria associated with the properties of smart contract platforms and criteria related to the development of smart contracts. It discusses properties like the support of programing languages, the basic execution mechanics, interoperability and availability of tools related to the development and test process.

Language Support & Community

One of the most commonly mentioned criteria in the analyzed evaluations is the language support, which describes the available languages in which smart contracts can be written and the characteristics of those languages [8].

The supported language plays an important role in the acceptance of a smart contract platform and therefore in the selection process. A broad language support and the support of common languages in general can make the integration in existing processes and tools easier and can help to leverage existing knowledge of developers and prevent the need of a steep learning curve [80].

Finally, the maturity of the supported languages should be analyzed as well, since a language or virtual machine that is still lacking in functionality, can cause problems or delays during the development process [80].

Another aspect when evaluating smart contract platforms in terms of the development process is the size & activity of the running and supporting community of developers, which is usually the first address to get help when problems occur [8].

Execution Model

The scope of the supported languages is a relevant and often cited factor in the comparison of smart contract platforms. Not all platforms support Turing-complete languages and are therefore independent of a particular application field [12]. Since this thesis deals with "all purpose" smart contract platforms it is valid to assume that both platforms' underlying mechanics and at least one supported language is Turing-complete.

Those underlying mechanics need to provide the three essential properties of smart contracts: determinism, isolation and termination [54]. Determinism is usually ensured by simply offering no non-deterministic functions or data sources and limiting or prohibiting dynamic calls. Isolation means that the executed smart contract needs to be sandboxed, so that it cannot influence other contracts or the system itself. Finally, termination in Turing-complete smart contract platforms is usually achieved by fee limits, calculation step limits or timers [11]. As mentioned, those properties can be ensured in different ways, which of course affects the possibilities of the platform and the smart contract that are developed on it.

Platform Interoperability

To achieve a higher degree of decentralization and distribution, "blockchain projects must be able to interact with one another. This is where interoperability comes into play" [79].

The isolation and the lack of communication between smart contract platforms is a recurring problem. This issue can cause obstacles when trying to distribute your DAPP to a broader audience and was also recognized by the EU, which stated in a report that "[. . .] for blockchain-based platforms to succeed they will need to be able to communicate and share data, a property that is usually referred to as interoperability" [59].

There are basically two ways to achieve interoperability between smart contract platforms. First, two or more platforms can agree on a trusted third-party authority to transfer information and digital assets. That trusted third-party could also be used to provide external information which could easy identity management. Second, smart contract platforms could also share information directly with each other leveraging trust generated by smart contracts [59]. The choice of scenario also depends on the overall strategy of the platform since it might include trusting a centralized authority (see Objectives & Strategy).

It is important to evaluate smart contract platforms based on their ability or strategy regarding interoperability, since it is a limiting factor in the development and distribution of smart contracts or DAPPs.

Identity Management

Another aspect of smart contract platforms, which is highly linked to platform interoperability in general, is the management of digital identities.

"A digital identity is defined as information on an entity used by computer systems to represent an external agent that may be a person, organization, application, or device" [16].

One of the key characteristics of existing blockchain platforms is pseudo-anonymity provided by using arbitrary addresses which are not linked to real identities. But when we consider blockchains as smart contract platforms, we also have to acknowledge that some applications have the need for accountability and transparency in terms of the identity of the user and therefore for a secure association with real-life identities. This need may derive from different directions, but most likely from legal obligations. Especially when you consider the rise of security tokens [16].

Therefore, analyzing a smart contract platform's ability to incorporate digital identities is a crucial part of the selection process.

Tool Support

As smart contract platforms become increasingly popular, the need for effective programming in that area is increasing as well. Since smart contracts often deal with digital assets or cryptocurrencies themselves, it should be clear that an effective and secure development process is crucial [80]. But most current software development tools are not tuned for smart contract development and are therefore inadequate for the development of such applications. This means that there is the need of improved and specialized tools like customized IDE support, debuggers and testing tools [14].

According to a survey conducted in [14] the most needed tools for the development in the area of blockchains and smart contracts are tools that support testing. This includes for example the availability of testnets or the simulation of such. Furthermore, tools enabling static code analysis, formal verification and penetration testing are required. Tools easing the development process are required as well. This includes specialized IDEs with debugging support, error highlighting or easy deployment to test nets.

The availability of adequate tools is another factor which should be considered when selecting a smart contract platform, since the lack of tools might result in higher development costs and a higher number of bugs.

Application Standards

Smart contract standards improve the interoperability, re-usability and security of token and other types smart contracts and are therefore another crucial aspect when it comes to evaluating smart contract platforms.

Smart contracts in Ethereum can be based on application-level standards which are defined in Ethereum Request for Comments (ERC). The most widely used and common example being the ERC-20 Standard, which describes a simple token standard and focuses on the re-usability in different applications [57]. Some other smart contract platforms, like ICON, tend to build on ERC-standards as well, because of their popularity [55].

In contrast to that, NEOs uses it's on scheme. In NEO token or smart contract standards in general are described as "NEO Enhancement Protocol" (NEP). Therefore, comparing the scopes of the different standard sets is necessary.

3.3.4 Execution and Operation

This section deals with the criteria that are relevant after a smart contract was deployed into a live system or mainnet. The execution of smart contracts and the operation of distributed applications depends on many aspects like the block time, the throughput of transactions and the associated costs.

Block Time

The block time describes the time between the creation of two subsequent blocks. This setting is usually a matter of direct or indirect configuration. In networks which use Proof-Of-Work as a consensus protocol, the difficulty of the puzzle can be adjusted to reach a certain block time (see chapter 2.1) [4].

Since the block time depends mostly on the network's configuration rather than the smart contract platform itself, it usually differs between different public and private instances or networks [8].

The block time plays a crucial role in the throughput of a smart contract platform and therefore it is fair to assume that minimizing the block time to be goal of smart contract platforms. Unfortunately, a smaller block time yields a higher likelihood of blocks being created approximately at the same time and therefore the creation of stale blocks. A block is considered stale when it was successfully created and verified by a subset of the network but is eventually discarded when another longer chain achieves dominance. Smaller intervals therefore increase the latency between the submission and confirmation of new blocks, since forks are more likely. This is one of the reasons why Bitcoin uses a relatively long block time of ten minutes [102], [103].

Block Confirmation Time

The latency between submission and confirmation mentioned in the previous chapter is called block confirmation time. It describes the amount of time or the number of blocks it takes to confirm a block. If a block was not discarded during this time, it is very unlikely that it will be eventually discarded in the future. For example, on Bitcoin there is a recommended block confirmation time of 1 hour or 6 blocks [102].

Since blocks are created in a distributed manner, two or more blocks can be created at the same time. Those blocks may contain conflicting transactions and therefore the design of a distributed application has to consider the fact, that even though a transaction was part of a block, it might be discarded later on. Otherwise this would open the door for double spending attacks - the very problem blockchains tried to solve in the beginning. The resolution of conflicting blocks or transaction is part of the consensus protocol of the blockchain [12].

This latency of course causes problems when it comes to real world processes, especially when assets with fast moving prices are involved. An example would be the trade of a security token. The time difference between submission and confirmation can pose a financial risk for the issuer.

As a consequence, the block confirmation time influences the application's design and surrounding processes and therefore needs to be considered when selecting a smart contract platform for an application.

Throughput and Scalability

Throughput in a smart contract platform describes the transactions processed per second. This criterion too is heavily influenced by the configuration of the particular network and therefore differs between public and private networks [4].

Many experts see the current limited throughput of existing platforms like Ethereum as the main issue smart contract platform face nowadays. The discussion about throughput is usually focused on the consensus protocol, which heavily influences the throughput of a platform. However, there are other factors influencing this key figure as well. For example, the configured block size or special mechanism like sharding have an influence as well.

The throughput of transactions is an important factor when evaluating smart contract platforms since it can be a limiting factor when scaling an application [23].

Execution Costs

Executing smart contract functions is usually not free. Most smart contract platforms use a fee model to compensate node operators for the computational resources and storage used by smart contracts. Furthermore, fees prevent applications from consuming all the available resources of a network by for example by running an infinite loop [12], [102].

The actual costs depend on the used platform and the performed actions. For example, in Ethereum an operation on permanent storage is much more expensive than a calculation operation [100].

Fees are paid in a form of the cryptocurrency that most smart contract platforms provide. The model how those cryptocurrencies work can be quite different. Usually the owner of a transaction can influence the costs of a transaction by changing certain parameters that e.g. affect the priority of the transaction within the transaction pool.

Another aspect that should be considered is the volatility of such cryptocurrencies in comparison to FIAT currencies. Most companies earn their income and pay taxes in FIAT currencies, therefore at some point a conversion between a FIAT currency and the platform's cryptocurrency is necessary. If the volatility is high and the currency subject to speculation, the FIAT amount required to pay for a transaction can be very volatile as well. This makes it hard to calculate future costs.

Therefore, the cost or fee model and the volatility of the digital currency needs to be considered when evaluating smart contract platforms.

CHAPTER 4

Comparison of Ethereum and NEO

4.1 Project

4.1.1 Objectives & Strategy

Ethereum's objectives were already introduced in the basics chapter. Ethereum belongs to the second generation blockchains, which means that it focuses on providing a platform for smart contracts and programable transactions. Ethereum as such aims to be a general-purpose protocol for decentralized applications. Therefore, no special use case or application domain was in mind when creating the platform [102], [99].

To support a wide range of applications, Ethereum integrated a Turing-complete virtual machine in its blockchain, which we will discuss in detail later. Ethereum's goal to be a versatile platform can be found in the core principles of Ethereum which were defined by Vitalik Buterin [99]:

1. Simplicity: The Ethereum protocol should be as simple as possible, even at the cost of inefficiencies.
2. Universality: Ethereum does not provide features, but a versatile platform which can be used to develop DAPPs.
3. Modularity: The protocol should be built modularly, so that local changes do not affect the entire platform and features can be re-used.
4. Agility: Details of the protocol may change over time if new tests are conducted or new opportunities are found.

5. Non-discrimination and Non-censorship: The protocol should not try to restrict or prevent specific categories of smart contracts or DAPPs.

The fifth principle ("Non-discrimination and Non-censorship") states that Ethereum will not try to actively restrict or prevent undesired applications [99]. In summary: Ethereum is not built for any particular application domain nor does it want to discriminate or censor any kind of application.

In Ethereum everybody can create accounts without revealing their real identity. But whenever a transaction is conducted from an account, its public address is visible to the network. Those transactions might be linked to compromise the anonymity of an entity [101]. Therefore, the Ethereum platform does not support anonymity, but pseudo-anonymity. Anonymity and privacy are important issues for Ethereum and its founder Vitalik Buterin, who is currently working on a solution to increase anonymity by introducing mixer contracts [52].

Like Ethereum, **NEO** provides a Turing-complete smart contract platform, which theoretically allows all kinds of applications. But NEO was built with a specific goal in mind: building a distributed network-based smart economy system [1]. In NEOs whitepaper this goal is defined as follows: "NEO is the use of blockchain technology and digital identity to digitize assets, the use of smart contracts for digital assets to be self-managed, to achieve "smart economy" with a distributed network" [70].

According to the official NEO website, a smart economy is comprised of three elements: Digital assets, digital identity and smart contracts [86]. The focus of the platform lies on digitized assets and digital identities to support the proof of ownership of those assets. Furthermore, smart contracts can be used to automize the management of digital assets [58], [89]. NEOs definition of digital assets and identities were already discussed in the basics chapter.

"The NEO platform allows for linking the physical asset with an equivalent and unique digital avatar on its network. NEO also supports the protection of assets. Those assets registered on its platform have a validated digital identity and are protected by law. Digital identity enables verifiable key information about participating individuals, organizations, and other entities that exist in the digital context" [89].

NEO'S approach to include digital identities by design also means that governments and regulatory bodies will likely have a link to NEO's network and the assets traded on it [58], [89], [31]. One of the reasons to include digital identities was that the Chinese government expressed concerns regarding the anonymity of blockchains in regard to ICOs [33]. This regulatory-compliance enabled by the integration of digital identities distinguishes NEO from most other smart contract platforms. As we already discussed in the basics chapter, the identity feature is mandatory for operators of certain network nodes as well.

[1]The objective to build a smart economy was introduced in 2017 when the platform was relaunched and renamed from "Antshares" to "Neo".

4.1.2 Project Status

Ethereum has a live production network (mainnet) that is launched since 2015 but is undergoing consistent upgrades. The current version of Ethereum is 1.x and although it already acts a platform for thousands of smart contracts, it should not be considered finished [93], [29]. Those ongoing upgrades are necessary because "the Ethereum mainnet, if left unchanged, would become very hard or impossible to use due to severe performance degradation and increased storage requirements" [42].

There are currently three widely used Ethereum test networks (testnets) which are operated by community projects. They can be used by developers to test their smart contracts or DAPPs before the go live. Those testnets have the advantage that, like the mainnet, they are publicly accessible, but the execution does not cost any "real" Ether – Ether in the test networks is free to obtain. As of August 2019, the "Ropsten" testnet resembles the current mainnet's configuration best. The other two testnets ("RinkeBy" and "Kovan") both use a different consensus mechanism – Proof-of-Authority [13].

Ethereum is in comparison to other blockchain platforms well specified. The key documents and specifications of Ethereum are:

- The whitepaper from Vitalik Buterin ([99]), which specifies the goal and philosophy of Ethereum and the basic mechanics.
- The "Design Rationale" documentation, which explains the reasoning behind the basic technologies and algorithms ([37]).
- A yellow paper which defines and formalizes Ethereum's state transition function, virtual machine, algorithms and cost structure ([100]).
- Several API and console application specifications, so that applications and users can interact with Ethereum ([39], [40], [41]).

Besides that there is the official website of the Ethereum Foundation, which lists many resources on how Ethereum works and how to learn to write smart contracts or DAPPs with Ethereum ([38]). Two of the most used sources are EthHub ([48]) and eth.wiki ([50]), which provide detailed information on the Ethereum platform and the project itself, like roadmaps.

To summarize: Ethereum has stable live and test environments, that will evolve over time to keep it stable. It is already used by thousands of smart contracts and DAPPs. Furthermore, there is comprehensive documentation and specification available and a lot of further information sources which are created and maintained by Ethereum's community.

NEO also provides a functional live and test environment. The mainnet was launched in 2016 and the current version of the NEO platform is 2.x. Version 3.0 is already in development, which is currently planned to be released in 2020 [106]. Like Ethereum, NEO is under ongoing development and its mainnet is already used by live applications and can be considered stable.

The testnet is maintained by the NEO Project itself and they plan to introduce a second testnet which will be running using the new version NEO 3.0.

The official NEO project website offers a whitepaper ([70]) describing the general orientation and mission statement of NEO and a dedicated whitepaper describing the design of the smart contract platform ([72]). Those whitepapers give an overview of the basic design guidelines and the used technology and algorithms but provide no formal or more detailed specification.

There is currently a yellow paper similar to Ethereum's in development by a community project called "NEO Research" ([74]), which aims to provide a detailed specification by the end of 2019. Unfortunately, the project does not seem to be very active. As of August 2019, the last update was five months old, and the only finished chapter is the one about the consensus mechanism. All the other chapters are either blank or described superficially.

There is an API documentation available which shows how to interact with NEO nodes and therefore the network: [67].

NEO has developed an active community as well. The most active international community is called "City of Zion" ([20]), which works on documenting NEO and on several tools like wallets and block explorers. Furthermore, it provides several channels for NEO related news: [21], [66]. Other communities like neofans.org are quite active as well, but provide information in Chinese only, because NEO is still a project heavily driven from China as we will see in the next chapter.

To summarize: NEO provides a stable main- and testnet, which are continuously improved. In 2020 a new major version is planned to be released. Although there are some resources available, a detailed and formal specification of NEO and its virtual machine is missing. Furthermore, some communities decided to interact in Chinese, which could pose a barrier for developers from other regions.

4.1.3 Origin and Organization

In 2014, Vitalik Buterin and his co-founders created a Swiss company called **Ethereum** Switzerland GmbH to start the development of the Ethereum platform. But after the decision that the project would proceed as a non-profit, the Ethereum Foundation was founded – again located in Switzerland [47].

The development of Ethereum was funded using a crowdsale in 2015, which allowed participants to buy Ether using bitcoins. The sale lasted for 42 days and provided the Ethereum Switzerland GmbH with funds worth about 18 million dollars. Furthermore, 12 million Ether was retained in a dedicated development fund of the Ethereum Foundation [47].

The mission of the Ethereum Foundation is to "promote and support Ethereum platform and base layer research, development and education to bring decentralized protocols and tools to the world that empower developers to produce next generation decentralized applications (dapps)" [46]. As such they sign responsible for the development of the Ethereum client "Geth", solidity and widely used development tools. Furthermore, it spun off companies which focus on the development of the core implementations.

The Ethereum ecosystems consists of many different companies that participate in the development of Ethereum, infrastructure projects and DAPPs. Two organization with a high impact on Ethereum are ConsenSys and the Enterprise Ethereum Alliance.

ConsenSys was founded by one of the co-founders of Ethereum, Joseph Lubin. The organization acts as an incubator for Ethereum related projects like DAPPs, but also for infrastructure focused projects like MetaMask. In that role ConsenSys tries to facilitate communication and knowledge transfer between developers of DAPPs while allowing them to work autonomously [53]. Besides developer knowledge, ConsenSys – more precisely ConsenSys Labs – is also acting as a venture capital company by providing financial support for blockchain related projects. Furthermore, ConsenSys provides services for companies who want to incorporate Ethereum or want to create dedicated (private) Ethereum networks [53].

"ConsenSys's importance to Ethereum goes well beyond providing infrastructure tools. Positioning ConsenSys as the industry friendly face of decentralization gives the firm a large amount of influence in defining to organizations both what it is and how it can be useful. It also increases the chances of Ethereum adoption amongst corporations, as competitor protocols lack a similarly well-organized cheerleader body" [53].

Because it ties to decentralized projects and companies implementing Ethereum, ConsenSys has a huge influence on the Ethereum project itself. Furthermore, it maintains close relationships to government bodies like the European Commission or the South African Reserve Bank, which also benefits the entire Ethereum ecosystem.

ConsenSys is one of the co-founders and one of the leading companies behind the Enterprise Ethereum Alliance (EEA). The EEA wants to foster adoptions of Ethereum in various industries by customizing Ethereum to enterprises' needs. Its mission statement lists the following goals:

- "Deliver an open, standards-based architecture and specification to accelerate the adoption of Enterprise Ethereum.
- Create world-class Enterprise Ethereum Client Specifications and testing and certification programs that ensure interoperability, multiple vendors of choice, and lower costs for its members." [3]

Members of the Enterprise Ethereum Alliance are for example Microsoft, British Petroleum, JP Morgan, Accenture and Intel.

Antshares – which is now referred to as **NEO** 1.0 – was started by CEO Da Hongfei and CTO Erik Zhang, who already founded the blockchain company OnChain together. The research for NEO started in 2014. In 2015 a crowdsale that lasted 10 days was conducted and raised about USD 500 000. A second crowdsale raised 4.5 Million USD [86]. Those funds are used to develop the NEO platform.

In 2017 Anshares was rebranded to NEO and with that the focus was laid on creating a distributed system for the "smart economy". This new version is called NEO 2.0 and as of August 2019 is the most current version.

The development of the NEO protocol is steered by the NEO Foundation, a non-profit organization located in China. The Foundation is co-chaired by Da Hongfei and Erik Zhang, who have executive decision-making power for the NEO platform and can ultimately decide on its future [65].

Two organizations were founded during the restructuring of NEO's organization in 2018: NEO Global Development (NGD) and NEO Global Capital (NGC).

NGD is a sub organization of the NEO Foundation and focuses on research & development, marketing and community development [65]. NGC is a Singapore-based organization licensed for fund management. It will act "as the investment platform to empower high-quality blockchain projects" [65].

The company most associated with NEO is OnChain – the blockchain company that was founded by Da Hongfei and Zhang. The connection is so close, that Da Hongfei even needed to clarify publicly that NEO and OnChain are in fact separate companies: "First, I need to clarify that NEO and Onchain are separate entities, so Onchain doesn't own NEO, or NEO, Onchain." [77]

OnChain has its own blockchain product called Distributed Network Architecture (DNA), which helps other companies to set up blockchains. DNA is very similar to NEO and therefore profits from its ongoing development. Da Hongfei even mentions that interoperability between NEO and DNA-based chains will be possible in the future [77].

OnChain closely works with the Chinese government, which plans to incorporate blockchain in their systems [84]. The impact of the Chinese government on the development of NEO is unfortunately unclear: "We don't comment on any question that hints at a relationship between governments and NEO unless there is something concrete" said Da Hongfei [75]. But what is known is that NEO wants to enable the creation of legally compliant smart contracts to accomplish its goal to create a smart economy. OnChain contributes to the NEO project what ConsenSys contributes to Ethereum: maintaining a close relationship to enterprises and governments. Although in case of OnChain, it is mostly the Chinese government.

4.1.4 Governance

Ethereum uses an off-chain governance process, which means that the rules are not coded in the platform but applied on a social level. The governance process is based on documents called Ethereum Improvement Proposals (EIPs). EIPs are design documents that either present information to the community or describe a new feature of Ethereum or its surrounding processes. The following classification is applied to EIPs:

- "Standard Track EIPs" describe changes which will affect all or most Ethereum implementations, such as changes in the block validation ruleset or new application standards.
- "Meta EIPs" propose changes to processes connected to Ethereum, but which do not directly affect the Ethereum protocol. Examples would be changes concerning the decision-making process or changes to tools.
- Informational EIPs provide general guidelines, information or discuss design issues. They do not propose a new feature and do not require the community to reach consensus [9].

EIPs can be submitted by anyone using a Github project ([43]) and they should facilitate a discussion so that the community can reach consensus about the EIP. Finally, when no more discussion or adaptions are necessary, EIPs are approved [9]. But "in order for a Standard track EIP to actually be implemented, several other things must happen. After being accepted and merged, it may be discussed in the All Core Devs meetings if someone (such as the author) is willing to champion that EIP. If there is consensus among the core developers that it should be implemented, it's marked as Final and implementation work can begin" [6].

Although the process involves the Ethereum community, it is criticized for being too centralized since the core developers are eventually determining what changes are implemented in the core protocol [91].

NEO utilizes off-chain and on-chain governance. NEO provides two types of native tokens: NEO and NeoGas. The NEO-Token represents the right to manage the network and therefore participate in the on-chain governance process. According to the whitepaper token-holders can vote for representatives in the consensus protocol called bookkeepers and on certain network parameters. The role of bookkeepers will be discussed in a later chapter. Unfortunately, neither the whitepaper nor the yellow paper specifies the concrete network parameters on which the token holders can vote. The NEO-Token are not dividable but can be transferred and entitle the owner to a certain amount of GAS which will be added to the account of the holder similar to dividends in the equity market [70].

Off-chain governance is steered by the NEO Council, which consists of the founding members of the NEO Project. It is separated in three committees:

- The management committee handles strategic decisions.
- The technical committee deals with technical decisions.

- And the secretariat is responsible for specific implementations [70].

New ideas can be discussed in NEO Enhancement Proposals (NEP), which work similar to EIPs in Ethereum. Like in Ethereum there are there categories of NEPs:

- Standard Track NEPs describe changes to the NEO protocol and its core components.
- Meta NEP describe processes around NEO and propose changes to those processes.
- Informational NEPs consist of design issues or present information to the community. They do not propose a new feature [107].

The final decision on which proposals are included in the protocol are made by Hongfei and Zhang – the founders of OnChain. Hongfei said in an interview: "Usually it's me and Erik making the decisions" [31]. This process is of course controversial. A member of the City of Zion, Ethan Fast, said that NEO runs as a "benevolent oligarchy" [31]. Which is a similar claim that the core developers of Ethereum have to face.

The way how governance in NEO is handled is likely to change with the new version of NEO which is planned to be rolled out in 2020. Erik Zhang, co-founder of NEO, says: "We will be actively collaborating with experts from the academia, industry and community to explore various governance mechanisms including liquid democracy, futarchy and some others emerged in recent times. [. . .] NEPs regarding on-chain governance changes will be published if satisfying outcomes are achieved after extensive research and simulation" [106].

4.2 Blockchain Properties

4.2.1 Public Chain and Dependencies

Ethereum offers a public network since 2015 and as of August 2019 has processed over 520 million transactions since its launch [44]. The network is permissionless, meaning that everybody can download a client and access the network.

Furthermore, the nodes implementing the Ethereum protocol represent a dedicated network, which does not rely on other blockchain network as infrastructure – therefore there are no direct dependencies to other blockchain networks [8].

NEO also features a public network. NEO's main network went live in 2016 and till August 2019 more than 24 million transactions were conducted [22]. The public network can be accessed without any permissions and multiple clients – for developers and regular users – are publicly available [69].

NEOs network does not rely on any other blockchain as infrastructure and is therefore, like Ethereum, an independent network.

4.2.2 Supported Deployment Types

As clarified in the last chapter, **Ethereum** maintains a public network – the mainnet. But it is also possible to create a separate network by running nodes in a different configuration (e.g. the network id).

Developers can create their own network and configure it so that it is basically another public Ethereum network, where every participant can read and write. This can be achieved by not restricting access to your nodes and using a consensus protocol that allows every node to create blocks – like Proof-of-Work.

When a consortium chain is required, which means that the permission to create new blocks is restricted to certain nodes, it is advised to use a different consensus mechanism. Usually Proof-of-Authority is used for that approach which can limit the right to create blocks to certain nodes. To enable other external nodes to connect to your nodes and read data, your nodes should be publicly accessible [36].

A private network is different from a consortium network such as that it does not allow the public to connect to the network and read data. This can be achieved by either restricting the access on the network layer – e.g. with firewalls – or by simply defining the peer nodes in the network configuration [36].

Ethereum therefore allows you to create dedicated networks for your company or applications, where read and write access can be adapted to your needs. Since the Ethereum implementation is open source, customizations on the protocol can be conducted by adding a layer on top of the existing protocol or by changing the existing implementation [36].

We already established that **NEO** provides a network which is public. And like Ethereum, NEO also allows its software to be run in a dedicated network. To create a dedicated network one has to install the NEO client software, but define a different "magic" parameter, which identifies the network. As an alternative one can just use pre-configured virtual machines or containers provided by communities like City of Zion.

Using that approach it is possible to create a dedicated public network. But since the creation of blocks or the connected voting is bound to the distribution of NEO-Tokens, those tokens need to be distributed to allow external people to be part of the consensus protocol.

It is also possible to create private and consortium chains using NEO. As already mentioned, the possibility to participate in the consensus process and therefore in the creation of new blocks, depends on the distribution of NEO tokens. If those tokens are not distributed publicly after the creation of the network, a private or consortium network can be achieved.

4.2.3 Consensus Protocol

The current version of **Ethereum**, which is 1.x, uses a Proof-of-Work based consensus protocol for its public chain. The algorithm used is called "Ethash" and is well defined in Ethereum's yellow paper ([100]). The next version of Ethereum (2.0), which will be gradually rolled out from 2019 to 2022, will include a change to a Proof-of-Stake based consensus protocol.

The current algorithm, Proof-Of-Work, uses a cryptographically secure nonce as proof that a certain amount of calculation was conducted [100]. The process of finding a fitting nonce is called "mining" and the nonce itself is used to calculate a hash value of the new block.

The resulting hash value has to be below a certain threshold, which is defined using the difficulty variable. The higher the difficulty value is, the harder it is to find a valid nonce. The difficulty is used to adjust the time needed to mine a block and is also a part of the block validation function. The difficulty is adjusted after each block depending on the time needed to calculate the previous block so that the target interval is maintained.

The relationship between the nonce and the difficulty is expressed in the following formula from Ethereum's yellow paper:

$$n \leq \frac{2^{256}}{H_d} \tag{4.1}$$

Where n is the nonce and H_d is the difficulty variable.

Ethereum's consensus protocol does not select a particular node to calculate the next block, but let's every mining node compete with each other. The first one to find a solution can add their block to the chain and receives a reward in Ether. As already mentioned in the basics chapter, it is possible that two or more nodes find a valid nonce at roughly the same time and the solution of one node could not be propagated to the other successful nodes before they found their solution. This scenario causes stale blocks or "uncles" as they are called in Ethereum. All but one path have to be dismissed eventually.

To define which block is allowed to remain, Ethereum uses a simplified protocol based on the GHOST protocol. If multiple paths exist, the one with the most calculation done can prevail. The path with the most calculation done, which is also called the "heaviest path" can simply be determined by summing up all the difficulties of the blocks in the path as described in the formulas below [100]:

$$\begin{aligned} B_t &= B'_t + B_d \\ B' &= P(B_H) \end{aligned} \tag{4.2}$$

With B_t being the desired total difficulty of block B and B' being its parent.

Another concern of Proof-of-Work algorithms are Application Specific Integrated Circuits (ASICS). ASICs for mining is basically hardware, which is specifically designed to calculate the proof needed for Proof-of-Work, giving the owner an advantage over miners with regular hardware. This of course diminishes the idea of decentralization. Ethereum addresses this issue by making the calculation of the nonce requiring a lot of memory and bandwidth so that the parallel calculation of multiple value is not feasible [100].

Stale blocks and ASICSs are not the only issue caused by Proof-of-Work based consensus protocols. Since hundreds of mining nodes are competing with each other, the process causes a high power consumption. In 2018 mining for blocks in Ethereum consumed as much power as Iceland. A single transaction consumes more electricity than an average US household per day [51]. "That's just a huge waste of resources, even if you don't believe that pollution and carbon dioxide are an issue. There are real consumers—real people—whose need for electricity is being displaced by this stuff" says Vitalik Buterin [51].

Therefore, Ethereum plans to switch to another type of consensus algorithm in Ethereum 2.0: Proof-of-Stake. "Proof of Stake (PoS) is a category of consensus algorithms for public blockchains that depend on a validator's economic stake in the network" [45].

So instead of using the calculation power of millions of CPU, the creator of the next block is chosen randomly based on their Ether balance. The participants of this process are called "validators" – as opposed to miners in Proof-of-Work. Those validators are responsible for proposing and voting on new blocks [45]. The list of validators is maintained by the upcoming Beacon Chain (the first phase of Ethereum 2.0) and nodes who want to be validators, and therefore be considered for the creation of blocks, need to lock up at least 32 Ether in a special contract [34]. Furthermore, validators can have different statuses, but only those who are marked as active can take part in the process. If one validator is caught cheating, they can lose their locked up Ether [51].

Estimations are that the change of consensus algorithm can save up to 99 % of the electricity consumed by Ethereum. Vitalik Buterin says further: "the PoW part is the one that's consuming these huge amounts of electricity. The blockchain transactions themselves are not super computationally intensive. It's just verifying digital signatures" [51]. The reduced costs of energy will also affect the reward mechanism. Since validators in Proof-of-Stake will have much less costs than miners in Proof-of-Work, the reward will by reduced as well.

To make the shift from Proof-of-Work to Proof-of-Stake a high priority in the project, the core developers of Ethereum added an exponential rise of the mining difficulty to the code – the "Difficulty Bomb". The effect is that the block time will increase over a few years till the entire mining process comes to a halt. This should motivate the community to push the Proof-of-Stake adaption and roll-out. However, when it was clear that the first targeted timeline would not hold, the difficulty bomb was "snoozed" by resetting the difficulty increase [51].

The Proof-of-Stake algorithm in Ethereum works as follows: The Ethereum blockchain keeps track of the validators. Everybody can become a validator by locking up an amount of Ether in a deposit by conducting a special type of transaction.

A new block can be created in each "slot", which a fixed time period. A pseudo-random algorithm then selects one of the validators for the calculation of the next block [45] – this role is called the proposer. The probability of one validator to be chosen depends on their deposited stake. The proposer has "the right to create a single block, and this block must point to some previous block (normally the block at the end of the previously longest chain), and so over time most blocks converge into a single constantly growing chain" [45]. The proposer basically collects all the votes from the rest of the validators, forms a block and publishes the result.

The acceptance of the block depends on the validators that vote on the validity of the published block. A block is accepted if 2/3 of the validators vote positively on the next block [28].

Validators can withdraw from the list of validators anytime, but their stakes will continue to be locked up for a certain amount of time (currently 97 days [34]). After this period, the stake including added rewards and potentially deducted penalties will be returned.

The Transition from Proof-of-Work to Proof-of-Stake will start in 2019 when the first phase of Ethereum 2.0, which is called "Beacon Chain" will be rolled out. The Beacon Chain will run alongside the current Proof-of-Work Chain to ensure continuity [28].

NEO uses a different algorithm to reach consensus: Delegated Byzantine Fault Tolerance (dBFT). The mechanism is based on real-time voting for bookkeepers who are responsible for proposing and validating new blocks. This approach is similar to Ethereum's Proof-of-Stake mechanism, but claims to be far more efficient since only small subset of nodes is actually involved in the validation process – the delegates or bookkeepers.

As a consequence, there are two types of nodes in the NEO network:

- Ordinary nodes who keep a copy of the ledger and can add new transactions to the pool.
- Bookkeeping nodes who are additionally responsible for proposing and validating new blocks. Furthermore, bookkeepers need to have their identity revealed and verified [71].

As mentioned before voting is an integral part of the consensus protocol and is part of the on-chain governance process described earlier. Users who own NEO tokens, which in general give you the right to manage the network, can vote for bookkeepers. Those bookkeepers act as delegates in the consensus process [71].

dBFT guarantees safety and liveliness as long as the number of erroneous bookkeeping nodes is not higher than a third of the total bookkeeping nodes (F):

$$F = \lfloor \frac{(N-1)}{3} \rfloor \tag{4.3}$$

With N being the number of bookkeeping nodes.

Consensus is reached if the number of nodes that accept a block is equal or greater to the safety level M.

$$M = N - F \tag{4.4}$$

Each consensus round and its included data is called a “View”. “If consensus cannot be reached within the current View, a View Change will be required. We identify each View with a number v, starting from 0 and it may increase till achieving the consensus” [71]. One bookkeeping node is selected in each View to propose a new block – this role is called “speaker”. Each bookkeeping node has a unique number p assigned (0 to N-1) which is used to identify the node. The selection of the node is based on the block height h and the number of the current View v:

$$p = (h - v) \mod N \tag{4.5}$$

"A round of consensus consists of 4 steps [. . .]:

- Speaker starts consensus by broadcasting a Prepare Request message.
- Delegates broadcast Prepare Response after receiving the Prepare Request message.
- Validators broadcast Commit after receiving enough Prepare Response messages.
- Validators produce & broadcast a new block after receiving enough Commit messages” [71].

If not enough nodes answer during a certain time frame – M nodes are required - a view change is triggered.

The process allows to have a single block finality, meaning that no forks – like in Ethereum’s Proof-of-Work consensus protocol – are necessary [71]. As a result, when a transaction is part of a new block, it cannot be reverted later on – like in Ethereum currently. Therefore users do not have to wait for a certain number of blocks to confirm a transaction [24]. As already described, this performance advantage is achieved by reducing the degree of decentralization. This tradeoff fits the general trilemma of blockchains which says that “blockchains can generally have only two of the following three properties: scalability (that is, performance in terms of speed and volume), decentralisation or security” [59].

It should also be considered that as of September 2019 five out of seven nodes are operated by the NEO Foundation itself. Another one is operated by City of Zion, which is financially dependent on the NEO Foundation. The last node is operated by KPN, a Dutch telecom provider, making only one node independent from the NEO Foundation.

The decentralization of bookkeeping nodes was planned for 2019, but was eventually delayed to the migration effort of NEO 3.0: “Since there is a migration required for NEO 3.0, the decentralization plan is currently on hold in order to implement the migration plan more efficiently.” a spokesman of the NEO Foundation said [31].

4.3 Platform and Development

4.3.1 Language Support & Community

Ethereum as smart contract platform is based on the Ethereum Virtual Machine (EVM) which executes EVM code – a Turing-complete stack-based bytecode language [8].

There are a few high-level languages, which can be used to develop smart contracts in Ethereum. Those languages are compiled into EVM code before they are stored in the blockchain.

Solidity is currently the most used programing language for the Ethereum platform. Solidity’s syntax is similar to C and JavaScript. Other languages, which are supported by the EVM include:

- LLL, which is a low-level Lisp-like language. It is still supported, but according to Vitalik Buterin developers are not encouraged to use it [17].
- Serpent, which is similar to Python, but already deprecated [18].
- Vyper, which is a strongly typed language based on Python.

As one can see, the language support is quite limited. Although the available languages are related to existing and well-known languages like JavaScript and Python, developers still have to learn a new language to be able to write smart contracts on Ethereum. That poses an entry barrier for new developers.

On the other hand, the community around Ethereum and solidity is quite active. In the time range of August 2018 to August 2019 Nearly 24.000 questions were tagged with “Ethereum” on the platform StackExchange – from which around 23.200 were answered by the community. When considering only questions that contain “Solidity” we see that there were about 2400 question in the same time range. Over 2300 of those questions were answered by the community [92].

“In order for a blockchain ecosystem to operate, nodes must execute transactions and smart contracts in a virtual machine. Ethereum 1.0’s virtual machine was called the Ethereum Virtual Machine (EVM). With the switch to Ethereum 2.0 and the Beacon Chain, the network’s virtual machine will be upgraded to eWASM, a virtual machine based off Web Assembly, which is defined by the World Wide Web Consortium (W3C) as an open-source standard. Because WASM supports a number of coding languages, eWASM could allow smart contracts written in any language to be executed on Ethereum, as opposed to just ones written in Solidity in today's EVM."[28]

Like Ethereum, **NEO**'s smart contract platform is based on a virtual machine. This virtual machine is called NeoVM and uses an own low-level language to execute smart contracts. The code of smart contracts is stored in that language on the NEO blockchain. One goal of NEO was to create a smart contract platform with the advantages of Ethereum's virtual machine, but without the limitation regarding higher level programing languages [11]. NEO provides a set of development tools called "DevPack", which allows developers to write smart contracts in existing languages and with existing IDEs using plug-ins. Those higher-level languages are then compiled into NeoVM code.

The following languages are currently planned to be supported by NEO:

- C#, VB.Net, F#
- Java, Kotlin
- Python
- GO
- JavaScript

The first batch of supported languages are .NET based: C#, VB.Net and F#. NEO already provides compilers and IDE plugins for those languages, which allow developers to use their familiar IDE [72]. As of September 2019, Java is not fully supported, but a compiler with "basic features" is available [63]. It is therefore advisable to use C# instead of Java.

"The NeoContract smart contract system is the biggest feature of the seamless integration of the existing developer ecosystem. Developers do not need to learn a new programming language but use C#, Java and other mainstream programming languages in their familiar IDE environments (Visual Studio, Eclipse, etc.) for smart contract development, debugging and compilation" [70].

Although NEO targets a broad developer audience, the community seems to be rather inactive with less than 100 entries on StackExchange during the last year. A similar number of entries can be found on SegmentFault.com, which is a Chinese developer platform like StackExchange.

4.3.2 Execution Model

Smart contracts in **Ethereum** are executed in a virtual machine called Ethereum Virtual Machine (EVM), which is based on a simple stack-based architecture. From a formal perspective, Ethereum is a transaction-based state machine. It takes the current state and uses transaction to modify it into some final state [100]. The first state, also called genesis state, is defined in the genesis block of Ethereum. “Transactions thus represent a valid arc between two states; the ‘valid’ part is important—there exist far more invalid state changes than valid state changes. Invalid state changes might, e.g., be things such as reducing an account balance without an equal and opposite increase elsewhere. A valid state transition is one which comes about through a transaction” [100]. Formally:

$$\sigma_{t+1} = \Upsilon(\sigma_t, T) \tag{4.6}$$

Where Υ is Ethereum state transition function and σ represents the state. T stands for a transaction. A transaction is considered valid when it passes the following tests:

1. The transaction has to be well formed
2. The transactions signature has to be valid
3. The transactions nonce has to be valid – meaning that it has to be equivalent to the sender account’s nonce
4. The gas limit is not allowed to be smaller than the intrinsic gas used by the transaction
5. The sender account balance has to contain at least the cost of the transaction [100].

The EVM supports Turing-complete languages. Therefore, smart contracts could trigger endless loops, which would eventually freeze the entire network, since they are executed in the context of blockchain transactions. Therefore, termination is a crucial property of smart contracts, which means that smart contracts must finish execution in a specified time limit [54].

Ethereum achieves termination by using a fee meter which relies on gas. “In order to avoid issues of network abuse and to sidestep the inevitable questions stemming from Turing completeness, all programmable computation in Ethereum is subject to fees. [...] Thus any given fragment of programmable computation (this includes creating contracts, making message calls, utilising and accessing account storage and executing operations on the virtual machine) has a universally agreed cost in terms of gas” [100]. When the gas used exceeds the pre-defined gas amount of the transaction (“gasLimit”) the execution is terminated and the changes undone. The used gas (no matter if the execution was terminated or not) is paid by the sender of the transaction. To obtain the Ether value of the gas used, the gas value has to be multiplied with the gasPrice, which is defined for each transaction individually. Because of the limit of computation, the EVM is called quasi-Turing-complete [100].

The EVM does not offer non-deterministic functions, since Ethereum is based on a distributed system where the execution must yield the same result on all nodes [54]. For example, it is not possible to create random numbers in smart contracts, because the number would be different on different nodes thus violating the determinism requirement. However, it is possible to use pseudo-random numbers which rely on data that is generally unknown at the time of transaction creation. That includes usually the block hash, the timestamp and the block's beneficiary address [100].

Furthermore, the data access is limited to the EVM and on-chain information. In general, smart contracts in Ethereum have access to three types of information:

- The stack, a last-in-first-out container to which values can be pushed and popped
- Memory, an infinitely expandable byte array
- The contract's long-term storage, a key/value store. Unlike stack and memory, which reset after computation ends, storage persists for the long term [99].

Finally, the virtual machine approach ensures that smart contracts are isolated from each other.

Smart contracts in **NEO** are executed in the NeoVM, a Turing-complete stack-based virtual machine. Its design resembles the Java Virtual Machine and .NET runtime and consists of three components: the execution engine, the stacks and the interoperation service layer. The "ExecutionEngine is the core of NeoVM [and it is] mainly responsible for loading scripts and executing corresponding instructions, such as flow control, stack operation, bit operation, arithmetic operation, logical operation, cryptography, etc." [68].

NeoVM uses four stacks for the execution of smart contracts:

- The InvocationStack stores execution contexts of different smart contracts.
- The EvaluationStack stores data which is used by the instructions in the execution process. Each execution context uses its own EvaluationStack.
- The AltStack stores temporary data for the execution. Each execution context has its own AltStack.
- The ResultStack stores the final execution results.

According to the NEOs whitepaper the NeoVM ensures consistent execution results on all nodes within the NEO network. To achieve this goal, the NeoVM does not offer any non-deterministic functions. An example is the retrieval of the system time, which is a common function in regular applications. However, this function is non-deterministic since it cannot be guaranteed that every node in the network will obtain the same value. Therefore NEO does not return the actual system time, but the creation time of the current block [72]. Another common functionality is randomness. True randomness would result in different values on different nodes, which would violate the determinism requirement. NEO therefore does not provide the generation of strong random numbers to ensure determinism – like Ethereum does. But pseudo-random numbers can be generated using the hash of the current block [72].

Another concern regarding determinism are external data sources. If data which is obtained during runtime is non-deterministic, then the entire function becomes non-deterministic. To ensure determinism, NEO only allows to access deterministic data sources. Those data sources are the blockchain ledger, including all blocks and transaction, and the contract storage space, which holds the values associated to the executing contract [72]. Data external to the NeoVM can only be accessed by using the integrated Interoperation Service Layer (ISL). If a contract needs to access other data, that data has to be transferred to the blockchain first – e.g. using oracles.

NEO allows smart contracts to call each other, but the targets have to be static: meaning the target cannot be specified at runtime, which allows the behavior of the smart contract call to be fully clear before execution [72]. NEO defines one limitation regarding calls: smart contracts cannot call each other recursively. Recursion is only possible within the boundaries of the contract [72].

The isolation of different smart contracts is ensured by the design of the virtual machine, which uses different execution contexts that are stored in the InvocationStack. This makes smart contracts sandboxed and the only way to access external data is to use the already mentioned ISL.

Furthermore, NEO uses a fee meter model similar to Ethereum. It is also based on gas and a pre-defined limit for each transaction. If the gas used exceeds that limit, the execution will be terminated, but the used gas won't be returned. GAS has to be paid for the creation and execution of smart contracts. This process eventually enables terminability of smart contracts. The difference to Ethereum is though, that gas does not have to be exchanged to the currency of the blockchain, since GAS is already a native token of the NEO blockchain [72].

The architecture of the NeoVM is illustrated in figure 4.1.

4.3.3 Platform Interoperability

Platform interoperability allows smart contracts or DAPPs to share information across different blockchains or networks. In case of **Ethereum** there is no built-in features to allow blockchain interoperability, but there are several projects aiming to close the gap.

A popular example of such projects is Chainlink. Chainlink is a decentralized oracle service which allows smart contracts to access off-chain data. Those oracles can serve as a bridge between different blockchains. Usually Oracles depend on trusted third-parties to provide information, which undermines the decentralization goal of blockchains.

In contrast, Chainlink enables the decentralized verification of information authenticity of data provided using oracles. This process is stake-based – Chainlink operators stake Chainlink tokens and can be penalized for providing wrong or erroneous data. On the

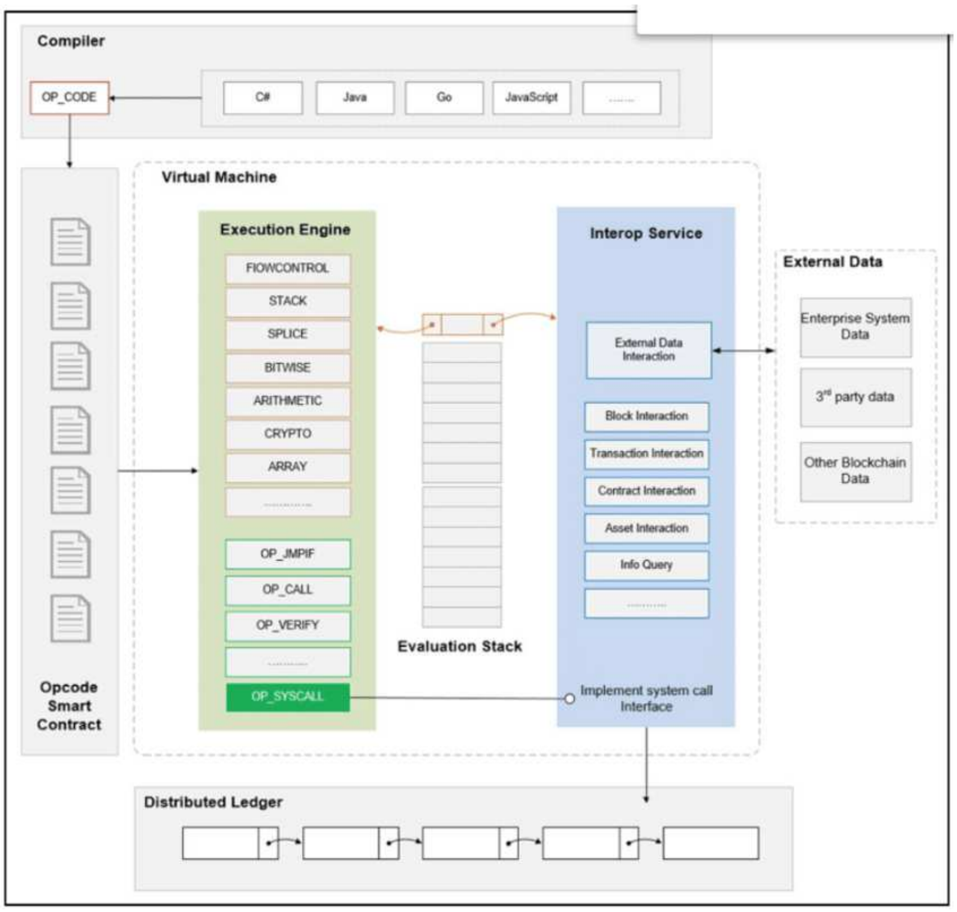

Figure 4.1: NeoVM Architecture [89]

other hand, those operators get rewarded with Chainlink tokens when they provide correct information. Since the entire process is decentralized, a consensus mechanism must be in place. In the case of Chainlink, there are two variations:

1. “Return to Mean — An average is reached by the nodes tasked with providing data to a smart contract. Nodes that provide information that does not fall within the margin of error or are penalized or not rewarded.

2. Democratic Consensus — The most common answer returned to a request for data is the one that is used. Nodes that provided data that does match that of the majority are penalized or not rewarded.” [56]

Chainlink gained popularity when they announced that they cooperate with SWIFT – the global interbank data transfer and payment system [78].

There are also specific solutions for blockchain interoperability between Ethereum-based networks. Like peaceBridge, which provides a mean to transfer tokens between Ethereum and Ethereum classic by facilitating smart contracts on both chains.

Although there is currently vast number of projects that aim to provide cross-blockchain interoperability, there is no accepted standard or widespread solution yet.

NEO follows a different strategy regarding cross-blockchain interoperability. NeoContract – the smart contract platform of NEO – wants to directly provide support for cross-chain interoperability. The reason is that NEO aims to have a consistent method on how digital assets are handled between different chains [72]. NEO's approach is described the protocol NeoX. NeoX does not only describe the exchange of digital assets between different blockchain networks but also defines how to execute cross-chain distributed transactions.

Cross-chain distributed transactions are transactions that are scattered across different blockchains. Keeping those transactions consistent across multiple chains is a challenge – cross-chain distributed transactions have to succeed or fail as a whole and not independently on each given chain [70]. As a consequence, NeoX could allow the development of cross-chain DAPPs, which interact with different blockchains. NeoX is based on atomic swaps, which allow users or smart contracts to trade assets which are located on different chains without the risk of one party not following the agreement.

Basically all NEO blockchains should be compatible with NeoX by default. That theoretically includes private networks based on NEO. Furthermore, blockchains developed by OnChain (based on DNA) are planned to be compatible as well [58] and theoretically every other platform could be made compatible too, as long as they provide some basic smart contract functions [70].

As promising as this approach sounds, as of September 2019 NeoX is not part of the live network nor is there a tutorial for development available.

4.3.4 Identity Management

Ethereum is designed to be a featureless platform. Therefore, Ethereum does not provide any out-of-the-box identity management features. However, there are ERC standards and DAPPs available, which deal with the creation of a decentralized identity management system. In contrast to existing identity management systems, those systems are not implemented using a centralized registry, but aim to give the individual the full control over their digital identity.

ERC-725 defines an identity standard that allows users to own and manage their own identity instead of giving the managing rights to centralized parties. Identities are identified by their public Ethereum address.

The most widespread application related to identity management on Ethereum is uPort. uPort provides a digital identity based on Ethereum, which can be identified using a public address. Identity ownership is therefore defined by the ownership of the corresponding private key. uPort furthermore, allows some flexibility, like key recovery methods, using Ethereum's smart contract capabilities.

uPort provides a registry, which is a shared smart contract that basically allows users to make statements about their identity. The data associated is not directly stored in the registry, but at the IPFS – the registry merely contains a pointer to that data [15].

uPort is used by the local government of Zug, a town in Switzerland, to allow its citizens to create a digital ID. Citizens of Zug can use their digital ID to access government eServices including voting [97]. Of course, the process of registration had to be verified by the local government using official documents. Nevertheless, this pilot project shows that decentralized digital identities can have real world applications, and might even be accepted by centralized institutions.

Digital identities is one of the three corner stones of **NEO**'s smart economy strategy ("Digital Assets + Digital Identity + Smart Contract = Smart Economy" [70]).

NEO's standpoint is that you need to have a digital identity to protect your assets and to allow government bodies to recognize you as the owner [58]. This position is in accordance with the Chinese government, which signed the "Digital Signature Act" in 2005 making digital signatures legally binding [98]. The first real-world application was released in 2016 when Microsoft China and OnChain created the "Legal Chain", which required a digital ID to be used [98].

NeoID, the digital identify feature integrated in NeoContract, will be part of the next version of NEO – NEO 3.0 – and is planned to be released in 2020. NeoID will allow to store information to individuals, organizations on the blockchain and is designed to be decentralized, which aligns with NEOs goal to give users and organization better control over their data [73]. NeoID is conceptionally structured into three parts: "Trust Model, Privacy Model and Game Model. The Trust Model describes the rules of trust in this distributed network. The Privacy Model describes the privacy protection scheme for users' online data. The Game Model describes the benefits and penalties of actions within the trust network. These three parts provide a mathematical model to abstract the real world, forming the basis of NeoID" [73].

Having your identity verified on the NEO chain can also have other implications. For example, when spam transaction from anonymous accounts occur. In this case the NEO bookkeepers might decide to prioritize transactions from verified accounts to keep the system running [86].

Although NeoID is not yet live or nor properly documented, NEO has made some advances regarding digital identities. In May 2019, Swisscom Blockchain released Seraph ID – a digital identity framework built on NEO. The goals of Seraph ID are similar to uPort. It wants to give users more control over their data and enable them to securely share their personal data [96].

4.3.5 Tool Support

As already mentioned in the chapter 3.3.3 the development tools which are needed the most are specialized IDEs or adequate plugins, debuggers and testing tools. Since transactions are executed in a distributed environment, tools to monitor the status of transactions are needed as well. To facilitate a comparison of Ethereum and NEO, the existence of such tools in both ecosystems will be analyzed.

Because of the large community of **Ethereum**, there is quite a large number of development tools available. One IDE specially created for Ethereum is Remix. Remix offers the possibility to create smart contracts in Solidity and Vyper and directly deploy them on the mainnet, testnet or a local test chain mock. Remix also provides an integrated debugger. It should be noted, that debugging a smart contract works different from debugging a local application because of its distributed nature. Debugging a smart contract usually involves executing the contract and then going through the execution log step-by-step.

Solidity, currently the most popular language for smart contract development on Ethereum, can also be written in popular IDEs like Visual Studio, Atom or IntelliJ through the use of add-ons [27]. Furthermore, there are other tools which can be used to debug contracts written in solidity like Truffle or the Ethereum Graph Debugger [27].

Since debugging involves the actual execution of the code, a test chain is needed during the development process. As already mentioned, there are multiple public test chains available for Ethereum. However, it can make sense to have a local chain for testing to ease the development process. In the Ethereum ecosystem it is quite simple simulate a local blockchain without the need to actually create a network. Two popular tools are the Remix IDE and Ganache.

There are also tools to ease the testing process. The Truffle Framework for example provides tools to write tests in JavaScript and automate runs during the build process. Again, those tools rely on test chains to be available. The testing of smart contracts should also include security tests, especially when they are part of financial applications. The DAO attack on the Ethereum blockchain shows how alleged small issues can have a huge financial consequence. There are tools for checking common vulnerabilities like reentrancy attacks or overflows in Ethereum smart contracts. One widely used is MythX, which analyzes Solidity and EVM code to detect vulnerabilities.

Last, block explorers like Etherscan are available for the mainnet as well as for testnets and allow developers to monitor transactions and check the current state of the blockchain.

There are similar tools in the NEO ecosystem. For example, NeoCompiler.io provides a web-based editor and compiler which allows to write smart contracts for NEO in C#, Python, Java and Golang. Furthermore, it provides an integrated test net, which makes deploying and testing contracts easy.

Since **NEO** allows to write smart contracts in existing languages, it focuses its effort on creating plugins for existing IDEs like Visual Studio and creating libraries for existing languages like Java. It should be noted that the documentation of NEO has inconsistencies regarding the supported languages and IDEs. For example, some sub-pages of the documentation list Java as supported, but other sub-pages recommend not to use the java compiler, because it is not yet finished: [70], [63].

The NeoVM provides a debugging option at the virtual machine level, where break points can be set on the contract code level [72]. This feature combined with the low coupling design should ease the debugging process in IDEs.

As mentioned before, NEO provides a public test net, which can be used to test and debug your application. There is also the possibility to create local networks or download ready-to-use containers. Furthermore, there is a community project called neo-local which shares the same goals as Ganache for Ethereum: providing a local simulated chain in a few clicks.

NEO provides a short tutorial on how to conduct tests with NEO contracts, but this is far from a complete test suit as provided in the Ethereum ecosystem. For example, there is no integrated test chain that makes sure that the tests start with a clean state. Furthermore, there are no tools available that focus on testing security aspects.

There are multiple block explorers available for NEO. Some of them like NEOSCAN.io are community driven. The basic functionality does not differ from the block explorers available for Ethereum, but some additional features, like the code validation, is missing.

4.3.6 Application Standards

Application standards in **Ethereum** are described in ERC documents (Ethereum Request for Comment). Those standards are not just recognized by Ethereum, but also by other platforms, like ICON, as well [55].

The most used ERC standard is ERC-20. ERC-20 defines a simple standard interface for fungible tokens, which means that it can be used for tokens whose individual units are interchangeable. This standard recognized by most wallet applications and one reason for its success is that it was used for most Initial Coin Offerings (ICOs) in the past [57]. The scope of the standard is compared to other standards quite limited: it allows the transfer and approval of tokens.

ERC-721 is similar to ERC-20, but deals with non-fungible tokens – tokens which are not interchangeable. Therefore, actions on tokens, like a transfer, require a unique ID to identify tokens. Furthermore, ERC-721 incorporates “safe” transfer functions, which means that if the recipient of a token is a contract-controlled account, a certain function is called on the contract to make sure that it can actually handle ERC-721 tokens. Otherwise, it would be possible that the tokens sent are lost forever. A popular game leveraging the ERC-721 standard is CryptoKitties, which deals with collecting and breeding virtual cats. Each cat is a one-of-a-kind – they are non-fungible. As of December 2019, the volume of CryptoKitties transactions exceeded 27 million USD.

Another standard worth mentioning is ERC-223, which extends the ERC-20 standard by enabling safe transfers similar to the ERC-721 standard but is still backwards compatible to ERC-20. Furthermore, it needs half the gas ERC-20 does.

ERC-777 which is also based on ERC-20, extends ERC-20 by allowing to mint or burn tokens during its lifecycle. Furthermore, it allows to approve operators, who can transfer token on users' behalf and also implements a safeguard for transferring tokens to the wrong address.

The standards mentioned are just a subset of the token standards defined in ERC documents and of course there are ERC standards that are not token related. For example, ERC-1167 defines a standard for a simple and immutable proxy contract. Another example is ERC-1484, which allows to create a decentralized identity registry for other smart contracts.

The number of application standards in **NEO** is much smaller – there are currently only seven final NEPs in total.

NEP-5 defines a standard for fungible tokens and incorporates a safe transfer function similar to ERC-223. The check in this case is not based on a certain function call, but on the receiving contract being "payable", which means it is designed to receive assets [2]. One of the most noticeable projects building on NEP-5 is NEX, which is a decentralized exchange built on NEO.

Another standard, NEP-11, which is similar to NEP-5 but deals with non-fungible tokens. Again, this standard uses a safe transfer function, which makes it similar to ERC-721 [94].

4.4 Execution and Operation

4.4.1 Block Time

When we discuss the term "block time", we need to distinguish between the expected block time and the actual block time. Usually the actual block time diverges from the expected block time, because a Proof-of-Work based algorithm is used and the solving time of is to some degree random.

As already mentioned, **Ethereum** currently uses a Proof-of-Work based algorithm. Mining nodes therefore have to solve cryptographic puzzle, before a new block can be created. The difficulty of the puzzle is adjusted whenever the actual block time diverges from the expected block time – usually because of added or removed computation resources. The reason behind this is that miners should not be able to impact the security of the network by adding more computational power – the relation of the block time and forks was already discussed in the previous chapters.

If the block time of the current block was lower than ten seconds, the difficulty is increased. If the interval was greater or equal than 20 seconds, the difficulty will be decreased. If

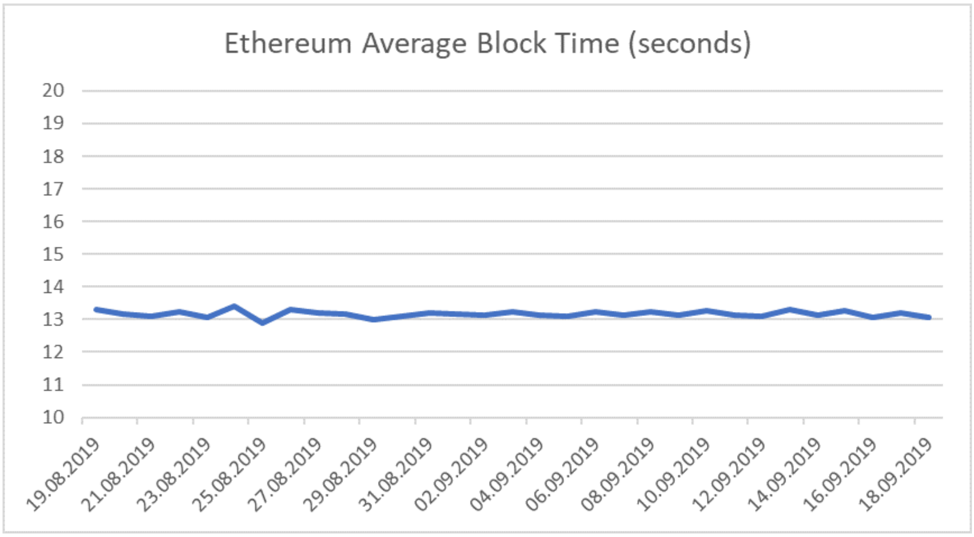

Figure 4.2: Average block time of Ethereum [44]

the block time was between ten and 20 seconds, the difficulty will not be adjusted [90]. Additional to the block time related adjustments, there is also the difficulty bomb, which is another factor that increases the difficulty.

Ethereum has an expected block time of approximately 14.5 seconds. That interval was chosen in respect to the size of the Ethereum network and the expected latency so that the number of forks stays low [90]. According to Etherscan.io, the actual block time of the last six month lies between 13 and 14 seconds [44]. Figure 4.2 illustrates the average block time between 19.08.2019 and 18.09.2019.

Although Ethereum's average block time is currently quite stable, it is still irregular. In contrast, Ethereum's new consensus algorithm, which is based on Proof-of-Stake, produces blocks more regularly.

This new algorithm is part of Ethereum's new release (Ethereum 2.0). Ethereum 2.0 will produce block regularly every 16 seconds. But the core developers are already discussing the possibility to reduce it to eight seconds [34].

NEO's consensus algorithm, Delegated Byzantine Fault Tolerance, is not work-based. Therefore, NEO provides a regular block time, like Ethereum's Proof-of-Stake algorithm. According the NEO's whitepaper the expected block time of NEO's mainnet is between 15 and 20 seconds [70].

The actual average block time of NEO lies between 16 and 17 seconds. Figure 4.3 shows the average block time between 19.08.2019 and 18.09.2019. There is currently no clear statement about the block time of NEO when it's new version (NEO 3.0) is released.

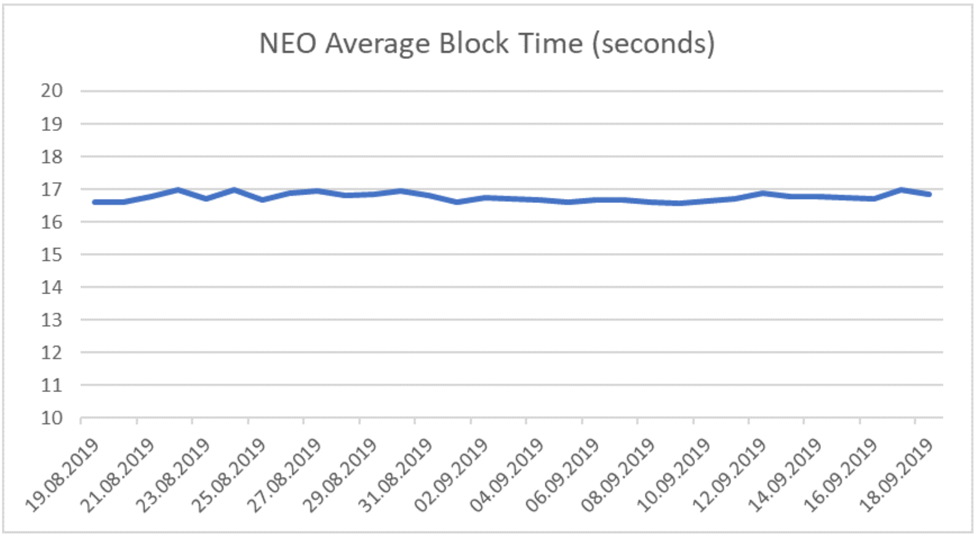

Figure 4.3: Average block time of NEO [76]

4.4.2 Block Confirmation Time

The block confirmation time describes the time (or blocks) needed before a transaction can be considered final – meaning it is impossible or very unlikely that its status is not going to change anymore. This concept is also known as "Finality". Although blockchains offer transaction immutability, they usually only offer a probabilistic transaction finality, meaning that transactions are not immediately final after becoming part of a block [87].

In case of **Ethereum**'s current version which is based on Proof-of-Work, it is generally recommended to wait 20 to 25 block confirmations to prevent double-spend attacks, which is equivalent to about six minutes [87]. Of course, this number is just a guideline and the actual time users should wait before they consider a transaction final depends on the value of the transaction – the higher the value, the more confirmed blocks users should wait before they consider a transaction final.

Transactions in Ethereum's current version cannot be considered final right away, because it is possible that the block containing the transaction is an uncle, meaning that multiple miners mined a block at roughly the same time, causing one of the blocks (or chain of blocks) to be stale [58]. From a model perspective, blocks in Ethereum can be seen as a tree, where multiple paths exist at the same time. The consensus mechanism is the algorithm that decides on the path through the tree, making all the siblings stale. The details were discussed in chapter 4.2.3.

Ethereum's next version, which will be based on Proof-of-Stake, provides "economic finality". Ethereum's Proof-of-Stake algorithm dictates that 2/3 of validators must stake Ether on the next block for it to become part of the blockchain [28]. "The only way that at any point in the future the canonical history will contain a conflicting block is if a large number of people are willing to burn very large amounts of money. If a node sees that this condition has been met for a given block, then they have a very economically strong assurance that that block will always be part of the canonical history that everyone agrees on" [19].

NEO is built upon a different kind of consensus algorithm: Delegated Byzantine Fault Tolerance. As already mentioned, a block can only be generated if enough bookkeepers vote for the block proposal. A bookkeeper cannot change their vote after the commit message was sent, which makes the vote final [71].

When a block was generated, it means that a sufficient number of nodes voted for the proposal and because of the requirements of NEO's consensus algorithm – roughly 2/3 of the nodes must vote for a block – this also means that there are not enough nodes left to successfully vote for any other proposal. Therefore the finality of a transaction can be given immediately after the block was generated [71]. There was only one case where the one block finality could not be achieved and that happened because of a network latency related bug [24].

As a consequence, the constant forking, which happens in Ethereum, cannot happen in NEO. Which also means that stale blocks cannot happen and a protocol to consolidate block trees like GHOST is not necessary. "Once a transaction is confirmed on the blockchain, it cannot be reversed or canceled. For financial applications, the finality of a transaction is a necessity." [73]

4.4.3 Throughput and Scalability

One of the major issues for blockchains is scalability. For example, **Ethereum** currently processes up to 15 transactions a second. On the other hand Visa's network processes up to 24.000 transactions a second [104]. The issue of scalability is part of the trilemma of blockchains, which means that blockchains can only have two of the following three properties: high scalability, high decentralization or high security.

Many private networks are able to achieve a much higher throughput and scalability due to a reduced degree of decentralization. Therefore, the issue of scalability usually affects permissionless networks, like the Ethereum or NEO mainnet.

As already mentioned, Ethereum can currently process up to 15 transactions per second. As of September 2019, the actual average number of transactions processed per second was 8.53 during the last 6 months (see Figure 4.4.

An obvious way to increase the throughput would be to allow more transactions per block [2] or to reduce the block time. However, increasing the block size would lead to a couple of issues, like higher network latency and higher storage requirements. Reducing the block time might actually increase the throughput and, as already mentioned, is part of the considerations of the Ethereum core developers once Ethereum uses Proof-of-Stake as consensus protocol.

[2]Actually, blocks in Ethereum are limited by the GasLimit (8.000.000 gas per block) and not by the number of transactions.

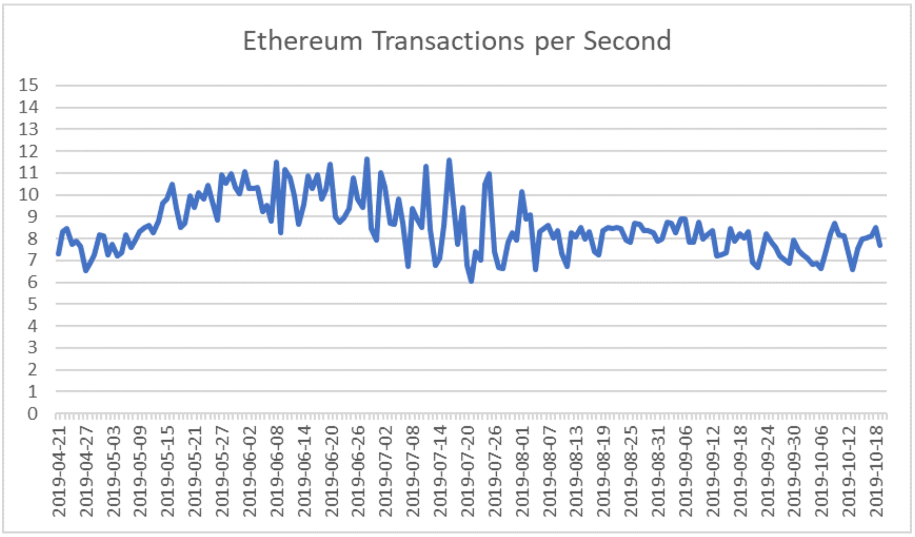

Figure 4.4: Ethereum Transactions per Second [5]

Ethereum 2.0 will also introduce sharding. At the moment, every node in the Ethereum network has to process every transaction in sequence. Although thousands of nodes are available, these transactions cannot be processed in parallel. Through sharding, the processing can be parallelized. Sharding partitions a larger blockchain into smaller ones called "shards". Each shard is represented by its own blockchain with its own state. The difference to just having multiple independent chains is that all shard-chains share a single Proof-of-Stake algorithm manifested in a central chain – the beacon chain. That means that there is a central pool of validators that validate transactions for all the shards [49].

The specification of Ethereum 2.0 states that the Beacon Chain will support up to 1024 shards, which theoretically allows to scale from 15 to about 15,000 transactions per second [28].

NEO claims that it surpasses the scalability issues Ethereum currently has and that it can provide up to 10.000 transactions per second [70]. But the actual throughput looks quite different. Currently NEO is processing less than one transaction per second on average with 4.25 transactions per second being the maximum value for a block between 19.09.2019 and 18.09.2019. The throughput is visualized in Figure 4.5. NEO is currently configured to include maximum 500 transaction per block and since the block time is configured to 15 seconds, the current maximum number of transactions per second is 33.

A community driven test conducted in 2018 showed that up to 2433 transactions per second could be processed [61]. However, the exact parameters of the tests are not known.

Furthermore, the NEO Foundation wants to incorporate sharding in their new version as well, so that scalability can be increased even further. NEO has an advantage in that area, because it already enforces static call relationships between smart contracts, which means

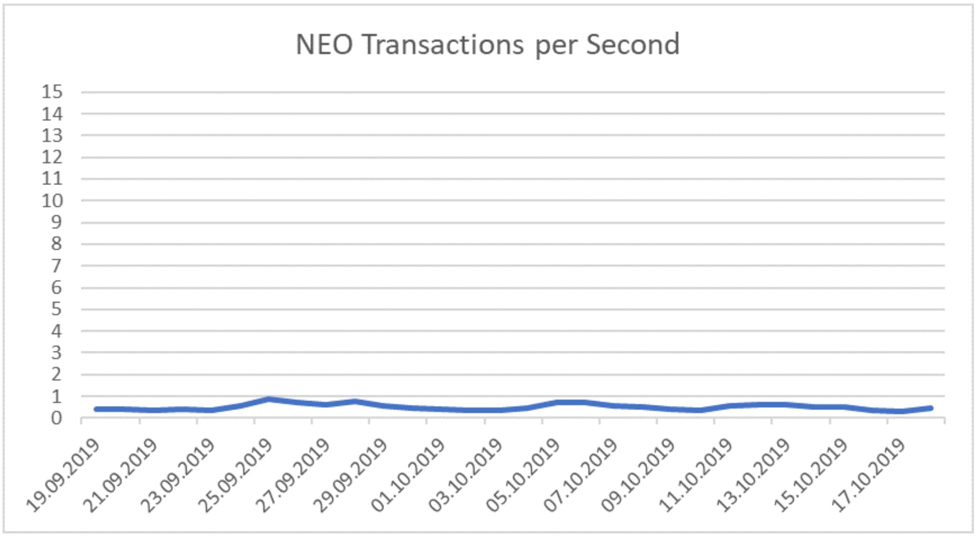

Figure 4.5: NEO Transactions per Seconds [22]

that the call targets needs to be specified before run time [72]. That restriction makes it easier to parallelize transactions. For example, contracts that modify the same state variable, would be identified and executed in a sequential manner whereby independent transactions could be executed in parallel.

4.4.4 Execution Costs

Every programmable computation in **Ethereum** is subject to fees. As already discussed, those fees prevent network abuse and are used as a compensation for miners. Every action like creating contracts, accessing storage or executing operations has an agreed cost which is given in the unit "gas". But there are exceptions: freeing up storage and self-destructing accounts does not cost gas but reduces the used gas of a transaction.

The sender of a transaction has to define a gas price. The gas price is exchange rate between gas and Ether, which is used to implicitly buy the needed gas using the sender's Ether balance at the beginning of each transaction.

All transactions are subject to a base fee of 21,000 gas (independent from its operations) and if the transaction includes the creation of a contract, 32,000 gas are charged additionally. One of the most expensive operations a transaction can include is to save values to the permanent storage – storing a 256 bit word costs 20,000 gas. That means that storing 1 kB of data sums up to 640,000 gas. The price of using the permanent storage was set so high, because permanent data has to be stored on every node in the Ethereum network. If storing data was cheap, maintaining an Ethereum node would require much more storage and therefore money, which would then decrease the decentralization of the network. The exact costs of operations in Ethereum can be found in Ethereum's yellow paper ([100]).

To make the operational costs of Ethereum as a smart contract platform tangible, I will demonstrate the costs based on an example: the creation of an ERC-223 token. Furthermore, I will translate the gas value into a value of a FIAT currency, which facilitates the comparison between Ethereum and NEO and also to classic systems. I will use US Dollar as the FIAT currency, since it is the most common in the cryptocurrency area.

The source code used to create the smart contract can be seen in Appendix A. By sending the creation transaction to the Ethereum testnet, we can observe that the creation costs about 936,000 gas. As already mentioned, the costs include the base fee and the creation fee. Senders of transactions also have to pay for every byte used in the transaction data and of course for the operations executed during the creation. Furthermore, the costs of the creation transaction depend on the size of the byte code of the smart contract.

To translate the gas value into a FIAT currency value, two factors are needed: 1. The gas price which can be chosen by the sender of the transaction. 2. The exchange rate between Ether and the selected FIAT currency – USD in our example.

As of 29.09.2019, the recommended gas price for a standard priority transaction is 10 Gwei [35]. 10 Gwei equals 0.00000001 Ether. The USD price of one Ether during the last year – as of 29.09.2019 – moved between USD 83.79 and USD 337.34 [25]. Those high and low prices are considered for the translation and will provide a price range for the creation of the contract.

Continuing the example from before, we can now calculate that the creation of a regular ERC-223 token on Ethereum would have cost between USD 0.78 and USD 3.16 during the last year.

NEO has a slightly different model than Ethereum. In NEO fees are also associated with transactions respectively with operations conducted in transaction and are defined in NeoGas (GAS). The difference is that GAS does not have to be translated into another cryptocurrency – GAS itself is a currency (see chapter 2.5).

Fees in NEO are divided into two parts: network fees and system fees. Network fees are fees which are paid to the bookkeeping nodes to pack the transaction into a block. Like the gas price in Ethereum, the amount can be adjusted by the sender of the transaction. The network fee has an influence on the priority of the transaction, since transaction with higher network fee per byte, will be preferred by bookkeeping nodes [62].

There is one type of transaction that does not require any network fee: claiming gas. As mentioned before, holders of NEO tokens receive a dividend-like reward in NeoGas. That reward needs to be claimed using a transaction [62].

System Fees, the second part of the NEO fee model, are defined by the actual resources consumed. The system fee cannot be adjusted by the sender of the transaction but is defined by the operations conducted within transaction. The NeoGas used for the system fee is distributed as the reward for NEO holders [62]. Claiming gas does not require a system fee and for all other types of transactions, the first ten GAS are free. That means that transaction that use ten or less GAS do not require a system fee.

Most of the operations cost between 0.001 and 0.1 GAS. But again, storing data permanently is compared to the other operations rather expensive: storing 1 kB of data costs 1 GAS [64]. This is due to the same reason as in Ethereum – storage needs to be provided on all nodes participating in the NEO network. The main difference between Ethereum's and NEO's fee model is the cost for creating a smart contract. Whereas in Ethereum it costs only of fraction of what storing 1 kB data costs, in NEO the costs are much higher. The costs of creating a smart contract in NEO oscillate between 100 and 1000 GAS [72].

The costs will be again illustrated using a common example. I will use the NEP-5 token standard (see chapter 4.3.6) as an example, because it is quite similar to the ERC-223 standard we used before. The source code can be found in Appendix B. Our token requires the ability to use storage, but not the ability to conduct dynamic calls, therefore the costs for creating our token are 490 GAS [3] . As of 29.09.2019, the USD price of one NeoGas moved between USD 1.06 and USD 7.03 during the last year. That means that creating a regular NEP-5 token would have costed between USD 519.40 and USD 3444.70.

The difference is also visible when comparing the costs of storing 1 kB of data to the permanent storage. Using the same method from before, we get costs between USD 0.54 and USD 2.16 for Ethereum and costs between USD 519.40 and USD 3444.70 for NEO.

Of course, those high costs hinder people from creating smart contracts, which is also acknowledged by the NEO Foundation: "Currently, the relatively high cost of deploying and running smart contracts leads to a reluctance in smart contract usage and development. [. . .] In NEO 3.0, we will address this issue by significantly reducing the deployment and execution costs of smart contracts [. . .]. Prior to the NEO 3.0 implementation, credible projects can apply for grants from the NEO Foundation with contract deployment costs." [73]

The current scenario does of course not facilitate decentralization, since only a few developers will want to afford those high costs. By giving grants to selected projects, the NEO Foundation can steer which kind of projects will be run on the NEO network, which of course can have positive effects for the project itself, but again diminishes the blockchains idea of decentralization.

[3]The costs of creating a smart contract that needs storage on NEO are fixed to 500 GAS, but the first 10 GAS are waived, so we end up with 490 GAS.

CHAPTER 5

Discussion

This chapter aims to discuss the key differences between Ethereum and NEO and the potential consequences. Furthermore, it deals with potential sustainability and viability issues of DAPPs developed on those platforms.

5.1 Key Differences

Ethereum and NEO are smart contract platforms with Turing-complete virtual machines at their core, which theoretically allows them to support all kinds of smart contracts respectively DAPPs. Therefore, both platforms have many properties in common. However, there are also some substantial differences, which we will discuss in this chapter.

As discussed in the "**Project**" section, Ethereum and NEO have different goals. Ethereum wants to be a featureless generic platform, that does not discriminate or censor DAPPS and their developers and supports the creation of decentralized applications or organizations. To achieve their goal, it is an important factor that Ethereum is not dependent on a single country or their government.

NEO on the other side has a different approach by setting their primary goal to be a platform for the "smart economy". To achieve this, NEO will incorporate features for identity management and cross-chain compatibility, which is in contrast to Ethereum's approach to be a featureless platform. Smart economies will probably have the need to work with existing government bodies, which NEO acknowledges by incorporating existing digital identity standards in their platform. The focus on smart economies is not necessarily a negative aspect for developers - it depends on the focus of the application. If you want to create a DAPP that deals with digital assets and digital identities, it could be worth considering a platform that focuses their development effort on those features. However, most of the features that should support the smart economy are not yet finished. This brings us to the next difference: the maturity of the platform and its documentation.

Ethereum is a tough benchmark when it comes to maturity. Although Ethereum's platform is still not in its final form and subject to rigorous changes, thousands of DAPPs already use the platform. The featureless platform was already enhanced by many projects that support features like identity management or cross-blockchain interoperability. Furthermore, there is a lot of documentation available for Ethereum including a yellow paper which describes Ethereum formally. NEO also provides a production network. However, it is not as widely used as Ethereum and most of the key features are still missing. This is especially a problem, because the platform and its current scope are not properly documented. The whitepaper is a mix of features of the current version (NEO 2.0) and the future version (NEO 3.0) and contains contradictory information. Furthermore, as of October 2019, NEO's yellow paper, which aims to define the technologies on a formal level, is mainly blank with only on section containing information. The low maturity of its documentation makes it hard for developers to actually use and trust NEO. However, larger projects will likely have the support – financial and knowledge wise – of the NEO foundation respectively OnChain. Smaller projects have to decide if they want to use a not properly documented platform, wait for the next version of NEO – which is hopefully better documented – or simply chose another platform.

Both platforms have an ecosystem of companies surrounding them that are the link to enterprise projects – see ConsenSys respectively OnChain. However, another aspect mentioned during this thesis is the connection between NEO and the Chinese government, which remains unclear to certain degree till today. The potential effects of this relationship will be discussed in the chapter 5.2.

When considering the **blockchain properties** of both platforms, there is one difference that stands out: the consensus protocol. Since this protocol eventually determines which changes are valid and part of the blockchain, and which ones are not, it is a crucial part of every smart contract platform. Ethereum currently uses a Proof-of-Work based protocol, in which every mining node can try to propose the next block. The sustainability issues that arise from such a protocol will be discussed in chapter 5.3. When Ethereum 2.0 will be released, a new consensus protocol that is based on Proof-of-Stake will be in place. Again, anybody can be part of the process and propose and validate blocks – the mirrors the goal of Ethereum to facilitate decentralization. But this high degree of decentralization comes with a cost. If we consider the trilemma of blockchains, we have to acknowledge that this high degree of decentralization comes with costs at either at the scalability or the security side of the platform.

NEO has set its focus on throughput and scalability and accepts a lower degree of decentralization to achieve its goal. NEOs consensus protocol uses the Delegated Byzantine Fault Tolerant (dBFT) algorithm, which essentially reduces the proposing and validation of new blocks to seven bookkeeping nodes. All other nodes can indirectly influence the block creation by voting for certain bookkeeping nodes – those nodes therefore will be fair and correct, otherwise they won't be elected again. This approach currently requires a large amount of trust from the users of NEO since five to six nodes are under the control of the NEO Foundation. This gives them the two-third majority needed to accept or reject blocks on their own. The founders of NEO claimed that this is going to change as soon as the development of NEO 3.0 is done, but again users will have to trust the NEO Foundation. As a developer of DAPPs using a public network, you have to decide which one is more important: decentralization or scalability. But one should also consider that the consensus protocol is not the only factor driving the throughput of transaction. Furthermore, the trilemma of blockchains merely states that if you increase one factor at least one of the other two will decrease. But you cannot measure the scalability of two platforms by comparing their degree of decentralization. I will discuss the performance implications later on in this chapter.

When we look at the "**Platform and Development**" section, we can see that Ethereum and NEO exhibit more differences. Ethereum has a very limited language support. Currently there are only two actively used programing languages for Ethereum and developers cannot use existing languages like Java or C# - they have to learn new ones. Ethereum claims that with Ethereum 2.0, this situation will improve, because the EVM will be migrated to eWASM, the Ethereum flavored WebAssembly. This allows Ethereum to support a wider range of languages and tools. Unfortunately, the new virtual machines is scheduled for a later version of Ethereum 2.0, which means that developers will have to wait until 2021. NEO on the other hand already supports multiple languages, like Java or .net based languages like C#. This is a huge advantage since developers can use the language they already know and do not have to learn no ones. Furthermore, it should be considered that most DAPPs consist of on-chain and off-chain parts, which means that projects will contain code for smart contracts and regular applications. Using the same programing languages for both parts eases the maintenance of the code and the use of tools. But one should keep in mind that some of the languages NEO lists as supported languages are in fact not yet fully supported. Again, it is tricky to determine the exact status of each compiler since documentation is sparse. If NEO does not speed the development of further compilers, Ethereum will eventually catch up in terms of language support, which will diminish one of the main advantages of NEO over Ethereum.

Another aspect that needs to be considered is the community of developers. Ethereum has a very active community, which means that help can be provided in many cases. NEOs community is much smaller and some parts of it interact with each other in Chinese, which makes it harder for non-Chinese developers to take part. One could argue, that the size of the community will increase over time, but it will be hard for NEO to build up an active community when the current price model only allows well-funded projects to use the platform. Private or semi-professional developers, who can create smart contracts on Ethereum quite cheaply, would have to spend thousands of dollars on NEO. Of course, this makes NEO for many developers unattractive, which as a result reduces the size of the community.

The outlook of both platforms regarding identity management and cross-platform interoperability is promising. There are currently DAPPs for both applications on Ethereum. uPort for example is the most popular DAPP regarding identity management and was already implemented by a local government in Switzerland. NEO also hosts a DAPP dealing with identity management from the Swiss telecom provider Swisscom. NEO also wants to incorporate both features – identity management and cross-chain interoperability – in its platform, but those features are not live yet. The cross-chain interoperability of NEO will play a huge role in the overall vision of OnChain, which we will discuss later on.

Ethereum and NEO also differ in terms of application standards. Ethereum offers a wide range of ERCs for tokens, identity management, proxy contracts and so on. NEO offers a far more limited range of application standards and most of them are focused on tokens. This reflects the overall strategy of NEO, since digital assets are a corner stone of the proposed smart economy. Therefore, it is likely that the focus on token standards will continue. This illustrates why selecting a smart contract platform with a fitting strategy is important when creating DAPPs. NEO will most likely focus on standards that serve their goal of a smart economy – if your application fits the smart economy description, there will likely be standards fitting your needs.

Finally, we will discuss the differences of Ethereum and NEO in terms of the **execution and operation** of smart contracts. Both platforms have a block time of around 15 seconds, but this is likely to change with the new versions of the platforms – Ethereum 2.0 and NEO 3.0. The block confirmation time on the other hand is quite different. Whereas Ethereum's current block confirmation time is around six minutes, NEO offers instant finality. This is due to the different consensus protocol used. If your application requires instant confirmation, NEO has a clear advantage. This advantage will diminish once Ethereum 2.0 goes live, which will introduce a Proof-of-Stake based consensus protocol. So, if a project requires instant finality, NEO is currently the better option.

If smart contract platforms will ever be widely adopted, scalability will play an even more important role than it does now. If you look at the current actual throughput of both platforms, Ethereum provides a higher figure, but in theory NEO could achieve a much higher throughput. NEO's promise of 10,000 transactions per second has not yet been verified. As soon as there is the first productive DAPP on NEO which actually generates that amount of transactions, more projects might be persuaded to create DAPPS based on NEO. Again, if NEO wants to be a serious alternative to Ethereum, they should act rather quickly, because Ethereum will gain ground in terms of scalability by switching to Proof-of-Stake and by incorporating sharding in their new version.

Even with those changes in mind, the potential throughput of traditional system is still much higher –Visa's systems process up to 24,000 transactions per second [59]. Developers should therefore keep in mind that it is not only about choosing the right smart contract platform, but also about deciding if and which part of the application should be developed on blockchains at all.

If you look at the costs of creating and operating smart contracts, it is obvious that NEO is only suitable for larger projects that can afford thousands of US Dollars costs per smart contract. In contrast to that, Ethereum allows developers to create a contract for a couple of US Dollars. Of course, this seems like a huge disadvantage for NEO. But it also means that NEO is used by serious and well-funded projects only. It is therefore unlikely that a crypto kitten armada will clog the system and by giving out grants, NEO can chose the "right" projects in this first adoption phase. However, customer-facing projects could still encounter serious downsides when choosing NEO as their smart contract platform, since customers would encounter high fees as well, which would probably deter them from using the DAPP. Furthermore, the approach to compensate the high costs with grants diminishes the decentralization of NEO as a smart contract platform.

5.2 Applications and Markets

The question which platform to prefer also depends on the market your application targets. If China is a main target for you DAPP, NEO might be a good choice, because China is known for preferring their own solutions over global alternatives. Otherwise, Alibaba wouldn't close in on Amazon's market cap and WeChat would not dominate China's social scene [58]. Furthermore, OnChain, the enterprise facing company which works closely with the NEO Foundation, already creates blockchains for the Chinese government and Chinese companies. Those chains will eventually be compatible with NEO and therefore an entire blockchain ecosystem might emerge with NEO being the backbone of OnChain's blockchain concept. The plan is that NEO provides a public chain while OnChain provides private chains for enterprises and the ultimate goal is to link both worlds together [89]. If the plan succeeds, NEO is likely to play a major role in the Chinese blockchain market. Of course, this a strong argument for DAPP developers, who want to focus on the Chinese respectively Asian market, to use NEO.

Furthermore, one should not confuse the harsh stand of Chinas government against cryptocurrencies with the general rejection of blockchain technology – there are multiple government projects dealing with blockchains [89]. The degree to which NEO will play a role in Chinas blockchain plans is currently just speculation. So are the consequences of such a relationship. It might happen that Chinas interest in NEO goes hand in hand with stricter rules and higher demands. Tencent, the company which develops WeChat, serves as an example. When the Chinese government banned western social media platforms like facebook, they basically encouraged the Chinese population to use WeChat instead [58]. So, similar to the speculations around NEO, they prohibited a global solution and replaced it with a local one. Tencent on the other hand had to employ people which are close to the communist party. The consequences for a decentralized project, which wants to act globally, could be fatal.

NEO beats Ethereum currently in several aspects like theoretical scalability and language support, but Ethereum is an established platform with an active community. Furthermore, Ethereum is going to improve its performance and language support, which diminishes the advantages of NEO. Therefore, it will be hard for NEO to establish itself as a viable alternative to Ethereum – especially in the markets outside China respectively Asia. So, the question is if the NEO Foundation will actively focus on the Chinese market und enhance its cooperation with the Chinese government. There are nearly 1.4 billion people in China alone and, as mentioned before, Chinese have a tendency to prefer their own products over the global alternatives, this would favor a Chinese built blockchain technology [58]. Furthermore, changes in Chinas regulations can happen fast due to the political structure, which makes it even more important to have a close relationship to the Chinese government.

The close relationship between the NEO Foundation, OnChain and China does not mean that Ethereum based blockchains aren't an option for the Chinese government or for Chinese companies. The local government of Xiongan decided to use Ethereum instead of NEO for their blockchain projects [105]. To be more precise, the hired ConsenSys to implement an Ethereum based blockchain. This not only highlights the importance of ConsenSys for the Ethereum ecosystem, but also shows that it is possible for Ethereum to compete in the Chinese market.

The case, that NEO wants to focus on the Chinese or Asian market is just one valid scenario. NEO could also try to become a globally successful blockchain and according to their CEO Hongfei, that is the plan for now. The cooperation with Swisscom is a good example: Swisscom Blockchain decided to use NEO instead of Ethereum for their identity management application, because they saw a good fit of their requirements to NEO. To gain further interest from Chinese and non-Chinese companies, it would be beneficial to give up control of further bookkeeping nodes and thus increasing the decentralization and trust in the platform.

5.3 Sustainability

When we talk about the sustainability of smart contract platforms, we have to distinguish between environmental sustainability and the sustainability of projects in terms of their viability.

Environmental sustainability is mainly driven by the power consumption of the blockchain networks. Ethereum has currently a very high power consumption As already discussed, the Ethereum network consumed as much power as Iceland in 2018 – nearly the entire power was needed to run the Proof-of-Work based consensus algorithm [51]. This not due to a bug in the system but because the way how Proof-of-Work lets nodes compete against each other using computational resources. This was one reason for the decision to switch from Proof-of-Work to a Proof-of-Stake based consensus protocol. As already mentioned, the switch will happen with the new version of Ethereum.

NEO does already consume very little power. This has two reasons: First, NEO uses a consensus algorithm that is similar to Proof-of-Stake as it does not require nodes to compete using their computational power. Furthermore, the number of nodes that need to communicate and validate during the creation of a new block is very limited –

currently only seven nodes are involved. Second, the number of total nodes is very small in comparison to Ethereum, but if NEO is going to succeed in the global market, this number will likely increase.

The sustainability of projects basically means that the project is viable over a long period of time [59]. This is of course a key factor when selecting a smart contract platform.

Both Ethereum and NEO are open-source projects that were funded through ICOs. Furthermore, many technologies in both ecosystems are community-driven projects. That means that both projects depend on an active ecosystem of developers and other contributors to succeed [59]. As already mentioned, Ethereum has a much larger community than NEO. The community is also more international and exists for a longer time. This gives Ethereum an advantage over NEO.

Another aspect when it comes to sustainability is funding. Both projects are funded well through ICOs - Ethereum raised 18 million dollars during their ICO [47] and NEO raised 5 million [86]. Another aspect which is sometimes mentioned in connection with sustainability is the market capitalization of the related cryptocurrencies. This figure can be seen as an approximation of the interest in the blockchain and shows how easy it would be to obtain more funds [59].

Furthermore, dependencies on other companies or governments influence the sustainability as well. Ethereum is fairly independent from other companies or governments. The only dependencies that exists are to the core developers, who eventually decide about the development. NEO has a similar dependency to the founders of NEO, Da Hongfei and Erik Zhang. Unfortunately, the relationship between the NEO Foundation, OnChain and the Chinese government remains unclear. This could be a major issue in terms of sustainability. In the previous chapter we only considered the option that China favors NEO and incorporates it in its plans. However, it is also possible that the government wants to shut down the NEO Foundation or OnChain.

CHAPTER 6

Conclusion

This work compared Ethereum and NEO in regards to their properties as smart contract platforms. To achieve a structured comparison, a catalogue of criteria was derived based on existing comparisons and evaluations of smart contract platforms in scientific literature. The resulting catalogue was then applied to both platforms which required a detailed literature review of both Ethereum and NEO. As a final step, the gained information was summarized and interpreted, which allowed to answer the research questions of this thesis.

Although both platforms seem quite similar on a superficial level, the structured comparison revealed noteworthy differences. Both Ethereum and NEO are a blockchain-based smart contract platforms, which provide a Turing-complete virtual machine. This allows them to theoretically support all kind of smart contract applications. But a closer look at the overall strategies and goals reveals that the platforms want to achieve quite different goals: Ethereum favors decentralization but no particular application domain and wants to be an uncensored, featureless platform. NEO on the other hand wants to create a smart economy and therefore focuses on certain aspects like tokens, digital identity and cross-blockchain interoperability. Both platforms already host live applications, but again, considerable differences regarding the maturity could be identified. The different level of maturity can be observed for example in their documentations: whereas Ethereum has detailed documentation available, NEO does not even provide a proper specification of its virtual machine. On the other hand, NEO provides a much better language support and finality within one block – which are major concerns of smart contract developers. Unfortunately, NEO still has a small community of developers and demands high costs for the creation of smart contracts, which are also important factors for developers. One has to keep in mind that those differences might be temporarily, because both platforms are community-driven and will continuously improve. Ethereum's consensus algorithm for example will be changed from Proof-of-Work to Proof-of-Stake, which is much more similar the NEO's Delegated Byzantine Fault Tolerant algorithm.

In contrast to existing literature, this work used a structured approach to compare smart contract platforms instead of evaluating them individually. This was achieved by creating a catalogue of criteria, which was applied to both Ethereum and NEO, and allowed this thesis to explore potential issues which are not usually in the focus of literature – like the high costs of smart contract creation on NEO. Furthermore, the resulting catalogue can be adapted and reused for the structured comparison of other smart contract platforms as well.

This thesis additionally encompassed criteria, which were not yet considered in similar scientific literature. This includes the maturity of the documentation or the decentralization of decision-making nodes – both were especially relevant when evaluating NEO. Furthermore, evaluating the current status and roadmap of key features and the actual costs of the smart contract creation proved to be worthwhile. Another innovative part of this work was the subject itself. As already mentioned in the introduction, Ethereum is widely considered in the comparison of smart contract platforms, but NEO is not. By providing an in-depth comparison of Ethereum and NEO, this thesis closed this gap.

Starting from this thesis, there is room for future work concerning two directions. It was mentioned multiple times during this thesis, that both platforms are evolving – like most smart contract platforms do. Therefore, it would be worth evaluating how the future versions of both platforms compare and if they provide the features and performance gains they promised. Also, in the case of NEO, the course regarding the decentralization of decision-making nodes and the relationship to the government of China should be reevaluated in the future. The other direction of future work concerns the performance of NEO's mainnet. As NEO claims its throughput of transactions as one of its main advantages, the actual figures should be investigated. Getting an application that actually requires a high throughput on the NEO mainnet, could provide a meaningful comparison to Ethereum, but is currently difficult because of the high costs of operation.

NEO currently has some advantages over Ethereum in regard to throughput and language support, but with its new major version, Ethereum will eventually catch up. Therefore, if NEO cannot verify its claims regarding scalability, its role as globally used smart contract platform is dubious and developers need to ask themselves if the reduction of decentralization in NEOs consensus protocol is actually justified.

APPENDIX A

ERC-223 Token Source Code

```
pragma solidity ^0.5.2;
import './SafeMath.sol';

//ERC223 interface
contract IERC223 {
  function totalSupply() external view returns (uint256);
  function balanceOf(address who) external view returns (uint);
  function transfer(address to, uint value) external;
  function transfer(address to, uint value, bytes calldata data) external;
  event Transfer(address indexed from, address indexed to, uint value);
  event Transfer(address indexed from, address indexed to, uint value, bytes
      data);
}

//ERC223 Receiving Contract
contract ERC223ReceivingContract {
  function tokenFallback(address _from, uint _value, bytes memory _data)
      public;
}

/**
* @title Reference implementation of the ERC223 standard token.
*/
contract ERC223 is IERC223 {
  using SafeMath for uint;
  mapping(address => uint) _balances; // List of user balances.
  uint256 private _totalSupply;
  string public constant name = "Example Token";
  string public constant symbol = "EXAM";
  uint8 public constant decimals = 7;
  uint256 public constant INITIAL_SUPPLY = 1000 * (10 ** uint256(decimals));

  constructor() public {
    _mint(msg.sender, INITIAL_SUPPLY);
  }

  /**
  * @dev Total number of tokens in existence
  */
  function totalSupply() public view returns (uint256) {
    return _totalSupply;
  }

  /**
  * @dev Transfer the specified amount of tokens to the specified address.
  *      Invokes the `tokenFallback` function if the recipient is a contract.
  *      The token transfer fails if the recipient is a contract
  *      but does not implement the `tokenFallback` function
  *      or the fallback function to receive funds.
  *
  * @param _to    Receiver address.
  * @param _value Amount of tokens that will be transferred.
  * @param _data  Transaction metadata.
  */
  function transfer(address _to, uint _value, bytes calldata _data) external
      {
    // Standard function transfer similar to ERC20 transfer with no _data .
    // Added due to backwards compatibility reasons .
    uint codeLength;
    assembly {
      // Retrieve the size of the code on target address, this needs assembly
      codeLength := extcodesize(_to)
    }

    _balances[msg.sender] = _balancesamsg.sender].sub(_value);
    _balances[_to] = _balances[_to].add(_value);
    if(codeLength>0) {
      ERC223ReceivingContract receiver = ERC223ReceivingContract(_to);
      receiver.tokenFallback(msg.sender, _value, _data);
    }
    emit Transfer(msg.sender, _to, _value, _data);
  }

  /**
  * @dev Transfer the specified amount of tokens to the specified address.
  *      This function works the same with the previous one
  *      but doesn't contain `_data` param.
  *      Added due to backwards compatibility reasons.
  *
  * @param _to    Receiver address.
  * @param _value Amount of tokens that will be transferred.
  */
  function transfer(address _to, uint _value) external {
    uint codeLength;
    bytes memory empty;
    assembly {
      // Retrieve the size of the code on target address, this needs assembly
      codeLength := extcodesize(_to)
    }
    _balances[msg.sender] = _balances[msg.sender].sub(_value);
    _balances[_to] = _balances[_to].add(_value);
    if(codeLength>0) {
      ERC223ReceivingContract receiver = ERC223ReceivingContract(_to);
      receiver.tokenFallback(msg.sender, _value, empty);
    }
    emit Transfer(msg.sender, _to, _value, empty);
  }

  /**
  * @dev Returns balance of the `_owner`.
  *
  * @param _owner   The address whose balance will be returned.
  * @return balance Balance of the `_owner`.
  */
  function balanceOf(address _owner) public view returns (uint balance) {
    return _balances[_owner];
  }

  /**
  * @dev Internal function that mints an amount of the token and assigns it
    to
  * an account. This encapsulates the modification of balances such that the
  * proper events are emitted.
  * @param account The account that will receive the created tokens.
  * @param value The amount that will be created.
  */
  function _mint(address account, uint256 value) internal {
    require(account != address(0));

    _totalSupply = _totalSupply.add(value);
    _balances[account] = _balances[account].add(value);
    emit Transfer(address(0), account, value);
  }

  /**
  * @dev Internal function that burns an amount of the token of a given
  * account.
  * @param account The account whose tokens will be burnt.
  * @param value The amount that will be burnt.
  */
  function _burn(address account, uint256 value) internal {
    require(account != address(0));

    _totalSupply = _totalSupply.sub(value);
    _balances[account] = _balances[account].sub(value);
    emit Transfer(account, address(0), value);
  }
}
```

APPENDIX B

NEP-5 Token Source Code

```
using Neo.SmartContract.Framework;
using Neo.SmartContract.Framework.Services.Neo;
using Neo.SmartContract.Framework.Services.System;
using System;
using System.ComponentModel;
using System.Numerics;

namespace NEP5 {
  public class NEP5 : SmartContract {
    [DisplayName("transfer")]
    public static event Action<byte[], byte[], BigInteger> Transferred;

    private static readonly byte[] Owner = "...".ToScriptHash(); //Owner
        Address

    public static object Main(string method, object[] args) {
      if (Runtime.Trigger == TriggerType.Verification) {
        return Runtime.CheckWitness(Owner);
      } else if (Runtime.Trigger == TriggerType.Application) {
        var callscript = ExecutionEngine.CallingScriptHash;

        if (method == "balanceOf") return BalanceOf((byte[])args[0]);

        if (method == "decimals") return Decimals();

        if (method == "name") return Name();

        if (method == "symbol") return Symbol();

        if (method == "supportedStandards") return SupportedStandards();

        if (method == "totalSupply") return TotalSupply();

        if (method == "transfer") return Transfer((byte[])args[0], (byte[])
            args[1], (BigInteger)args[2], callscript);
      }
      return false;
    }

    [DisplayName("balanceOf")]
    public static BigInteger BalanceOf(byte[] account) {
      if (account.Length != 20)
        throw new InvalidOperationException("The parameter account SHOULD be
            20-byte addresses.");
      StorageMap asset = Storage.CurrentContext.CreateMap(nameof(asset));
      return asset.Get(account).AsBigInteger();
    }

    [DisplayName("decimals")]
    public static byte Decimals() => 8;

    private static bool IsPayable(byte[] to) {
      var c = Blockchain.GetContract(to);
      return c == null || c.IsPayable;
    }

    [DisplayName("name")]
    public static string Name() => "Example Token"; //name of the token

    [DisplayName("symbol")]
    public static string Symbol() => "EXAM"; //symbol of the token

    [DisplayName("supportedStandards")]
    public static string[] SupportedStandards() => new string[] { "NEP-5", "
        NEP-7", "NEP-10" };

    [DisplayName("totalSupply")]
    public static BigInteger TotalSupply() {
      StorageMap contract = Storage.CurrentContext.CreateMap(nameof(contract)
          );
      return contract.Get("totalSupply").AsBigInteger();
    }

    #if DEBUG
    [DisplayName("transfer")] //Only for ABI file
    public static bool Transfer(byte[] from, byte[] to, BigInteger amount) =>
        true;
    #endif

    //Methods of actual execution
    private static bool Transfer(byte[] from, byte[] to, BigInteger amount,
        byte[] callscript) {
      //Check parameters
      if (from.Length != 20 || to.Length != 20)
        throw new InvalidOperationException("The parameters from and to
            SHOULD be 20-byte addresses.");
        if (amount <= 0)
          throw new InvalidOperationException("The parameter amount MUST be
              greater than 0.");
        if (!IsPayable(to))
          return false;
        if (!Runtime.CheckWitness(from) && from.AsBigInteger() != callscript.
            AsBigInteger())
          return false;
        StorageMap asset = Storage.CurrentContext.CreateMap(nameof(asset));
        var fromAmount = asset.Get(from).AsBigInteger();
        if (fromAmount < amount)
          return false;
        if (from == to)
          return true;

        //Reduce payer balances
        if (fromAmount == amount)
          asset.Delete(from);
        else
          asset.Put(from, fromAmount - amount);

        //Increase the payee balance
        var toAmount = asset.Get(to).AsBigInteger();
        asset.Put(to, toAmount + amount);

        Transferred(from, to, amount);
        return true;
      }
    }
}
```

List of Figures

2.1 Blockchain data structure 6
2.2 Transaction lifecycle [101] 7
2.3 Comparison of Proof-of-Work and Proof-of-Stake [1] 8
2.4 Number of DAPPs by Category 12
2.5 Smart Contracts in Ethereum [102] 14
2.6 dBFT proxy voting 16

3.1 Catalogue creation process 22
3.2 Governance Overview 25

4.1 NeoVM Architecture [89] 53
4.2 Average block time of Ethereum [44] 59
4.3 Average block time of NEO [76] 60
4.4 Ethereum Transactions per Second [5] 62
4.5 NEO Transactions per Seconds [22] 63

List of Tables

2.1 Distributed and decentralized applications 11

3.1 Criteria collected from literature 20
3.2 Deployment types 27

Bibliography

[1] 3iQ Research Group. Consensus Mechanisms Explained, 2018. URL: `https://3iq.ca/3iq-research-group/consensus-mechanisms/`.

[2] Tyler Adams, Luodanwg, Tanyuan, and Alan Fong. NEO NEP-5, 2017. URL: `https://github.com/neo-project/proposals/blob/master/nep-5.mediawiki`.

[3] Enterprise Ethereum Alliance. Enterprise Ethereum Alliance: About, 2019. URL: `https://entethalliance.org/about/`.

[4] Shikah J. Alsunaidi and Fahd A. Alhaidari. A survey of consensus algorithms for blockchain technology. In *2019 International Conference on Computer and Information Sciences, ICCIS 2019*, 2019. `doi:10.1109/ICCISci.2019.8716424`.

[5] Amberdata. Amberdata Statistics, 2019. URL: `https://amberdata.io/dashboards/transactions`.

[6] Andreas Antonopoulos and Gavin Wood. *Mastering Ethereum: Building Smart Contracts and DApps*. O'Reilly UK Ltd, 2018.

[7] Arati Baliga. Understanding Blockchain Consensus Models. Technical report, 2017. URL: `www.persistent.com`.

[8] Massimo Bartoletti and Livio Pompianu. An empirical analysis of smart contracts: platforms, applications, and design patterns. In *10323 LNCS, Financial Cryptography and Data Security*, pages 494–509. Springer, 2017. URL: `http://arxiv.org/abs/1703.06322`.

[9] Martin Becze and Hudson Jameson. EIP 1: EIP Purpose and Guidelines, 2015. URL: `https://eips.ethereum.org/EIPS/eip-1`.

[10] Billy Bambrough. China Could Be About To Throw Its Weight Behind Bitcoin, 2019. URL: `https://www.forbes.com/sites/billybambrough/2019/07/29/china-could-be-about-to-throw-its-weight-behind-bitcoin/`.

[11] Blockgeeks. A Deeper Look at Different Smart Contract Platforms, 2018. URL: https://blockgeeks.com/guides/different-smart-contract-platforms/.

[12] Thomas Bocek and Burkhard Stiller. Smart Contracts - Blockchains in the Wings. In Claudia Linnhoff-Popien, Ralf Schneider, and Michael Zaddach, editors, *Digital Marketplaces Unleashed*, pages 169–184. Springer Berlin Heidelberg, Berlin, Heidelberg, 2018. URL: https://doi.org/10.1007/978-3-662-49275-8{_}19, doi:10.1007/978-3-662-49275-8_19.

[13] Francis Boily. Explaining Ethereum Test Networks And All Their Differences, 2018. URL: https://medium.com/coinmonks/ethereum-test-networks-69a5463789be.

[14] Amiangshu Bosu, Anindya Iqbal, Rifat Shahriyar, and Partha Chakraborty. Understanding the motivations, challenges and needs of Blockchain software developers: a survey. *Empirical Software Engineering*, 2019. doi:10.1007/s10664-019-09708-7.

[15] Pelle Braendgaard. What is a uPort identity?, 2017. URL: https://medium.com/uport/what-is-a-uport-identity-b790b065809c.

[16] Francesco Buccafurri, Gianluca Lax, Antonia Russo, and Guillaume Zunino. Integrating Digital Identity and Blockchain. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018. doi:10.1007/978-3-030-02610-3_32.

[17] Vitalik Buterin. Reddit: Does LLL have a future?, 2016. URL: https://www.reddit.com/r/ethereum/comments/3tfg7i/does{_}lll{_}have{_}a{_}future/.

[18] Vitalik Buterin. Tweet about Serpent, 2017. URL: https://twitter.com/vitalikbuterin/status/886400133667201024?lang=en.

[19] Vitalik Buterin. Ethereum: Proof-of-Stake FAQs, 2019. URL: https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ.

[20] City of Zion. City Of Zion, 2019. URL: https://cityofzion.io/.

[21] City of Zion. Medium Channel: City of Zion, 2019. URL: https://medium.com/@cityofzion.

[22] City of Zion. NEO Scan, 2019. URL: https://neoscan.io/.

[23] Victor Clincy and Hossain Shahriar. Blockchain Development Platform Comparison. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, 2019. doi:10.1109/COMPSAC.2019.00142.

[24] Igor Coelho, Vitor Coelho, Peter Lin, and Erik Zhang. Community Yellow Paper: A Technical Specification for NEO Blockchain. Technical report, 2019. URL: https://neoresearch.io/assets/yellowpaper/yellow{_}paper.pdf.

[25] Coingecko. Coingecko - ETH prices. URL: https://www.coingecko.com/de/munze/ethereum/historical{_}data/usd?end{_}date=2019-09-29{&}start{_}date=2018-09-29{#}panel.

[26] CoinMarketCap. CoinMarketCap, 2019. URL: https://coinmarketcap.com/.

[27] ConsenSys. Ethereum Developer Tools List, 2019. URL: https://github.com/ConsenSys/ethereum-developer-tools-list{#}ides.

[28] ConsenSys. The Roadmap to Serenity, 2019. URL: https://media.consensys.net/the-roadmap-to-serenity-bc25d5807268.

[29] ConsenSys. The Timeline to Ethereum, 2019. URL: https://media.consensys.net/the-timeline-to-ethereum-d143ba1f6878.

[30] Michael Crosby, Nachiappan, Pradhan Pattanayak, Sanjeev Verma, and Vignesh Kalyanaraman. Blockchain Technology - BEYOND BITCOIN. *Berkley Engineering*, 2016. doi:10.1515/9783110488951.

[31] Leigh Cuen. NEO Releases Detailed Financials Ahead of Cryptocurrency Relaunch, 2019. URL: https://www.coindesk.com/neo-releases-detailed-financials-ahead-of-cryptocurrency-relaunch.

[32] Dappptotal.com. DAppTotal Analytics, 2019. URL: https://dapptotal.com/analytics.

[33] Hui Deng, Robin Hui Huang, and Qingran Wu. The Regulation of Initial Coin Offerings in China: Problems, Prognoses and Prospects. *European Business Organization Law Review*, 19(3):465–502, sep 2018. doi:10.1007/s40804-018-0118-2.

[34] Ben Edgington. State of Ethereum Protocol #2: The Beacon Chain, 2018. URL: https://media.consensys.net/state-of-ethereum-protocol-2-the-beacon-chain-c6b6a9a69129.

[35] ETH GAS STATION. ETH GAS STATION, 2019. URL: https://ethgasstation.info/.

[36] Ethereum Foundation. Consortium Chain Development, 2019. URL: https://github.com/ethereum/wiki/wiki/Consortium-Chain-Development.

[37] Ethereum Foundation. Ethereum: Design Reationale, 2019. URL: https://github.com/ethereum/wiki/wiki/Design-Rationale.

[38] Ethereum Foundation. Ethereum Foundation Website, 2019. URL: `https://www.ethereum.org/`.

[39] Ethereum Foundation. Ethereum: JavaScript API, 2019. URL: `https://github.com/ethereum/wiki/wiki/JavaScript-API`.

[40] Ethereum Foundation. Ethereum: JavaScript Console, 2019. URL: `https://github.com/ethereum/go-ethereum/wiki/JavaScript-Console{#}console-api`.

[41] Ethereum Foundation. Ethereum: JSON RPC, 2019. URL: `https://github.com/ethereum/wiki/wiki/JSON-RPC`.

[42] Ethereum Foundation. Ethereum Roadmap, 2019. URL: `https://docs.ethhub.io/ethereum-roadmap/ethereum-1.x/`.

[43] Ethereum Foundation. Github EIPs, 2019. URL: `https://github.com/ethereum/EIPs`.

[44] Etherscan.io. Ethereum Average Block Time Chart. URL: `https://etherscan.io/chart/blocktime`.

[45] EthHub. Ethereum Proof of Stake. URL: `https://docs.ethhub.io/ethereum-roadmap/ethereum-2.0/proof-of-stake/`.

[46] EthHub. Ethereum Foundation, 2019. URL: `https://docs.ethhub.io/ethereum-basics/ethereum-foundation/`.

[47] EthHub. Ethereum: History and Network Upgrades, 2019. URL: `https://docs.ethhub.io/ethereum-basics/history-and-forks/`.

[48] EthHub. EthHub Homepage, 2019. URL: `https://docs.ethhub.io/`.

[49] EthHub. EthHub: Sharding, 2019. URL: `https://docs.ethhub.io/ethereum-roadmap/ethereum-2.0/sharding/`.

[50] Eth.wiki. eth.wiki, 2019. URL: `https://eth.wiki/en/home`.

[51] Peter Fairley. Ethereum Plans to Cut Its Absurd Energy Consumption by 99 Percent. *IEEE Spectrum*, 2019. URL: `https://spectrum.ieee.org/computing/networks/ethereum-plans-to-cut-its-absurd-energy-consumption-by-99-percent`.

[52] Omar Faridi. Buterin Proposes Increasing Privacy on Ethereum Using 'Minimal Design Mixer'. URL: `https://www.cryptoglobe.com/latest/2019/05/buterin-proposes-increasing-privacy-on-ethereum-by-using-minimal-desig`

[53] FlatOutCrypto. ConsenSys: Understanding one of the most important firms in crypto, 2018. URL: https://hackernoon.com/consensys-understanding-one-of-the-most-important-firms-in-crypto-7e1d66533

[54] Christopher G. Harris. The Risks and Challenges of Implementing Ethereum Smart Contracts. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 104–107. IEEE, may 2019. URL: https://ieeexplore.ieee.org/document/8751493/, doi:10.1109/BLOC.2019.8751493.

[55] Icorating. Smart Contract Platforms Review. *Icorating*, pages 1–35, 2018.

[56] Ivoidwarranties. Chainlink — A Requirement for Cross-Blockchain Interoperability & More, 2019. URL: https://medium.com/@ivoidwarranties/chainlink-a-requirement-for-cross-blockchain-interoperability-more-f994b2ee

[57] Victor Lai and Kieran O'Day. ETHEREUM ERC TOKEN STANDARDS, 2018. URL: https://crushcrypto.com/ethereum-erc-token-standards/.

[58] Noam Levenson. NEO versus Ethereum, 2017. URL: https://hackernoon.com/neo-versus-ethereum-why-neo-might-be-2018s-strongest-cryptocurrency-799

[59] Tom Lyons, Ludovic Courcelas, and Ken Timsit. Scalability, interoperability and sustainability of blockchains. Technical report, European Union Blockchain Observatory & Forum, 2019.

[60] NEO. NEO Enhancement Proposals, 2019. URL: https://github.com/neo-project/proposals.

[61] NEO Foundation. NEO CLI Chain Stats, 2018. URL: https://coranos.github.io/neo/charts/neo-cli-chain-stats.html.

[62] NEO Foundation. NEO Charging Model, 2019. URL: https://docs.neo.org/docs/en-us/tooldev/concept/charging{_}model.html.

[63] NEO Foundation. NEO Documentation: How to use Java to write a NEO smart contract, 2019. URL: https://docs.neo.org/docs/en-us/sc/devenv/getting-started-java.html.

[64] NEO Foundation. NEO System Fees, 2019. URL: https://docs.neo.org/docs/en-us/sc/fees.html.

[65] NEO News Today. NEO release statement on organisation restructure, 2018. URL: https://neonewstoday.com/general/neo-release-statement-on-organisation-restructure/.

[66] NEO News Today. NEO News Today, 2019. URL: https://neonewstoday.com/.

[67] NEO Project. NEO API Reference, 2019. URL: https://docs.neo.org/docs/en-us/reference/rpc/latest-version/api.html.

[68] NEO Project. NEO Documentation - NEOVM, 2019. URL: https://docs.neo.org/docs/en-us/basic/technology/neovm.html.

[69] NEO Project. NEO Node Introduction, 2019. URL: https://docs.neo.org/docs/en-us/node/introduction.html.

[70] NEO Project. NEO Whitepaper, 2019. URL: https://docs.neo.org/docs/en-us/basic/whitepaper.html.

[71] NEO Project. NEO Whitepaper: Consensus Mechanism, 2019. URL: https://docs.neo.org/docs/en-us/basic/technology/dbft.html.

[72] NEO Project. NeoContract White Paper, 2019. URL: https://docs.neo.org/docs/en-us/basic/technology/neocontract.html.

[73] NEO Project. Roadmap of NEO 3.0 Development, 2019. URL: https://neo.org/blog/details/4141.

[74] NEO Research. NEO Research, 2019. URL: https://neoresearch.io/.

[75] Neo_council. Official AMA with Da Hongfei and Erik Zhang, 2018. URL: https://www.reddit.com/r/NEO/comments/93vz5g/official{_}ama{_}with{_}da{_}hongfei{_}and{_}erik{_}zhang{_}is/.

[76] Neodepot.org. NeoStats: Average Block Time, 2019. URL: https://neodepot.org/stats/average-block-time.

[77] OnChain. Interview with Da Hongfei, 2019. URL: https://www.youtube.com/watch?v=BcmoSp7bL7g.

[78] Stephen O'Neal. Blockchain Interoperability Explained, 2019. URL: https://cointelegraph.com/explained/blockchain-interoperability-explained.

[79] Corlynne O'Sullivan. Blockchain Interoperability — The Value of Cross Chain Technology for User Adoption., 2019. URL: https://blog.usejournal.com/blockchain-interoperability-the-value-of-cross-chain-technology-on-use

[80] Reza M. Parizi, Amritraj, and Ali Dehghantanha. Smart contract programming languages on blockchains: An empirical evaluation of usability and security. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018. doi:10.1007/978-3-319-94478-4_6.

[81] Julia Poenitzsch. What's the difference between Decentralized and Distributed ?, 2018. URL: https://medium.com/nakamo-to/whats-the-difference-between-decentralized-and-distributed-1b8de5e7f5a4.

[82] Wessel Reijers, Iris Wuisman, Morshed Mannan, Primavera De Filippi, Christopher Wray, Vienna Rae-Looi, Angela Cubillos Vélez, and Liav Orgad. Now the Code Runs Itself: On-Chain and Off-Chain Governance of Blockchain Technologies. *Topoi*, 2018. doi:10.1007/s11245-018-9626-5.

[83] Marten Risius and Kai Spohrer. A Blockchain Research Framework. *Business & Information Systems Engineering*, 59(6):385–409, dec 2017. URL: http://link.springer.com/10.1007/s12599-017-0506-0, doi:10.1007/s12599-017-0506-0.

[84] Ron See. The company behind NEO: Onchain and its ultimate plan — DNA, 2017. URL: https://hackernoon.com/neo-onchain-and-its-ultimate-plan-dna-4c33e9b6bfaa.

[85] Ameer Rosic. What is Ethereum?, 2017. URL: https://blockgeeks.com/guides/ethereum/.

[86] Ameer Rosic. What is Neo Blockchain?, 2017. URL: https://blockgeeks.com/guides/neo-blockchain/.

[87] Samparsky. Blockchain Finality- Proof of Work and Proof of Stake, 2018. URL: https://medium.com/coinmonks/blockchain-finality-pow-and-pos-35915a37c682.

[88] Chinmay Saraf and Siddharth Sabadra. Blockchain platforms: A compendium. In *2018 IEEE International Conference on Innovative Research and Development, ICIRD 2018*, 2018. doi:10.1109/ICIRD.2018.8376323.

[89] Shobhit Seth. Why NEO Can Do What No Other Cryptocurrency Can Do. URL: https://www.investopedia.com/tech/china-neo-cryptocurrency/.

[90] Prabath Siriwardena. The Mystery Behind Block Time, 2017. URL: https://medium.facilelogin.com/the-mystery-behind-block-time-63351e35603a.

[91] Michael Spencer. Ethereum's Governance isn't Decentralized, 2019. URL: https://medium.com/futuresin/ethereum-governance-isnt-decentralized-da449da430f4.

[92] Stack Exchange. Ethereum Stack Exchange Questions, 2019. URL: https://ethereum.stackexchange.com/questions.

[93] State of the DApps. State of the DApps - Stats, 2019. URL: https://www.stateofthedapps.com/stats.

[94] Joe Stewart, Shane Mann, and Wyatt Mufson. NEO NEP-11, 2018. URL: https://github.com/neo-project/proposals/blob/61e53a07a319c537c57c988cbcb216ea75de0f89/nep-11.mediawiki.

[95] Karim Sultan, Umar Ruhi, and Rubina Lakhani. Conceptualizing Blockchains: Characteristics & Applications. 2018. arXiv:1806.03693.

[96] Tokenpost. Swisscom Blockchain launches self-sovereign identity solution Seraph ID on NEO, 2019. URL: https://tokenpost.com/Swisscom-Blockchain-launches-self-sovereign-identity-solution-Seraph-I

[97] UPort. First official registration of a Zug citizen on Ethereum, 2017. URL: https://medium.com/uport/first-official-registration-of-a-zug-citizen-on-ethereum-3554b5c2c238.

[98] Chris Wheal. History of the NEO cryptocurrency. URL: https://dex.openledger.io/history-of-the-neo-cryptocurrency/.

[99] Ethereum wiki. Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform. Technical report, 2017. URL: https://github.com/ethereum/wiki/wiki/White-Paper.

[100] Gavin Wood. Ethereum: a secure decentralised generalised transaction ledger. Technical report, 2019. URL: https://ethereum.github.io/yellowpaper/paper.pdf.

[101] Xiwei Xu, Ingo Weber, and Mark Staples. *Architecture for Blockchain Applications*. Springer International Publishing, mar 2019. doi:10.1007/978-3-030-03035-3.

[102] Xiwei Xu, Ingo Weber, Mark Staples, Xiwei Xu, Ingo Weber, and Mark Staples. Existing Blockchain Platforms. In *Architecture for Blockchain Applications*. 2019. doi:10.1007/978-3-030-03035-3_2.

[103] Xiwei Xu, Ingo Weber, Mark Staples, Xiwei Xu, Ingo Weber, and Mark Staples. Varieties of Blockchains. In *Architecture for Blockchain Applications*. 2019. doi:10.1007/978-3-030-03035-3_3.

[104] Wenli Yang, Saurabh Garg, Ali Raza, David Herbert, and Byeong Kang. Blockchain: Trends and future. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018. doi:10.1007/978-3-319-97289-3_15.

[105] Joseph Young. China's Decision to Choose Ethereum Over its Own Blockchain NEO is MonumentalNo Title, 2018. URL: https://btcmanager.com/chinas-decision-to-choose-ethereum-over-its-own-blockchain-neo-is-monumenta
?q=/chinas-decision-to-choose-ethereum-over-its-own-blockchain-neo-is-monum
{&}.

[106] Erik Zhang. Roadmap of NEO 3.0 Development. URL: https://medium.com/neo-smart-economy/roadmap-of-neo-3-0-development-e2ae64edf226.

[107] Erik Zhang. NEP 1: NEP Purpose and Guidelines, 2017. URL: https://github.com/neo-project/proposals/blob/master/nep-1.mediawiki.

[108] Yuan Zhang. Blockchain. In *Encyclopedia of Wireless Networks*, pages 1–4. Springer International Publishing, Cham, 2019. URL: http://link.springer.com/10.1007/978-3-319-32903-1{_}171-1, doi:10.1007/978-3-319-32903-1_171-1.