UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Villem-Oskar Ossip

Ethereum blockchain and HyperLedger Burrow blockchain comparative analysis

Bachelor's Thesis (9 EAP)

Supervisor: Orlenys López Pintado, PhD

Tartu 2019

Ethereum blockchain and HyperLedger Burrow blockchain comparative analysis

Abstract:

This bachelor's thesis aims is to introduce the blockchain technology, provide a comparative analysis of two different platforms Ethereum and Hyperledger Burrow, by creating two decentralized applications, and finally analyses if it is practical to implement Hyperledger Burrow to more complex applications like Caterpillar. At the beginning a brief explanation of blockchain, distributed ledger technologies is given with other related terminology. Then it is explained why and what was the motivation of selecting these technologies. More is depth comparative analysis in conducted based on example applications, showing what are the main differences and similarities between Ethereum and Hyperledger Burrow.

Keywords:

Blockchain Technology, Ethereum, Hyperledger Burrow

CERCS: P170 - Computer science, numerical analysis, systems, control

Ethereumi blokiahela ja Hyperledger Burrow blokiahela võrdlev analüüs

Lühikokkuvõte:

Käesolevas bakalaureusetöös tutvustatakse plokiahela tehnoloogiat, võrreldakse kahte erinevat platvormi, Ethereum ja Hyperledger Burrow, luukse kaks detsentraalsed rakendust ning viimasena analüüsitakse, kas on praktiline rakendada Hyperledger Burrow platvormi keerukamatele rakendustele nagu näiteks Caterpillar. Töö raames tutvustatakse plokiahela ja hajusraamatu tehnoloogiaid. Järgnevalt selgitatakse miks ja mis põhjustel antud tehnoloogiad võrdleva analüüsi tegemiseks valiti. Analüüs põhineb kahe näidisrakenduse võrdlemisel, näidates mis on peamised erinevused ja sarnasused Ethereum ja Hyperledger Burrow vahel.

Võtmesõnad:

Plokiahela tehnoloogia, Ethereum, Hyperledger Burrow

CERCS: P170 - Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

1. Introduction

When creating a business process management system which involves two or more parties, then often trusted third parties are used (e.g. banking, healthcare, insurances, etc). This kind of approach introduces additional risks as all the transactions and data storage is done by one or a few trusted third parties meaning that transactions are often slower and more costly. These parties alone have to provide backup storage and safe data transaction. To enable business process to be performed with cheaper and faster transactions without trusted third parties, one of the possible solutions is to use an agreement that can be enforced through a blockchain (using smart contracts).

There are several different blockchain technologies that provide the usage of smart contracts allowing to exchange property, money, or anything of value in a transparent, conflict-free way while avoiding the services of third parties like Bitcoin[1], Ethereum[2], Hyperledger Burrow[3], etc.

Bitcoin [1] in the early days of its life did not support smart contracts, but as it has evolved, it has gained this functionality. However, it is not as flexible and programmable as it is on Ethereum. Bitcoin's smart contract language Script is limited to certain combinations of signature checks, hashlocks, and timelocks. Bitcoin does not use Turing complete smart contracts, but instead, it is using stack-based language which supports transactions of monetary value.

Ethereum [2] is one of the most well-known blockchains that uses smart contracts. Ethereum has its own Turing complete programming language called Solidity. This means that third parties can enter into a transparent agreement with each other allowing to transfer value between parties or held inside smart contracts .

These technologies have their own limits. Bitcoin is very limited and can not be used for higher level smart contracting and Ethereum requires porting libraries from other systems, is an open-source, public blockchain-based distributed computing platform, which does not offer complete privacy. On the other hand this is where Hyperledger Burrow comes in. It provides a modular blockchain client with a

[1] https://bitcoin.org/en/

[2] https://www.ethereum.org/

[3] https://www.hyperledger.org/projects/hyperledger-burrow

permissioned smart contract interpreter partially developed to the specification of the Ethereum Virtual Machine (EVM) [3]. Hyperledger Burrow is for developing private business blockchain technologies.

Example of a business process management system is Caterpillar, which is running on Ethereum, but as Ethereum is a public blockchain and business processes tend to hold fragile data, it would be more suitable for business to run on top of network that can support private channels. Given thesis is going to find out, if HyperLedger Burrow is a suitable solution to Caterpillar privacy issue.

To compare Ethereum and Hyperledger Burrow technologies, we create two different applications based on Ethereum and Hyperledger Burrow. Author is going to find solutions to problems that occur with the architectures and evaluate which of these blockchain technologies is more suitable for this particular use case. Additionally business process execution engine Caterpillar is introduced and briefly analyzed.

The first goal of this thesis to give an overview of Ethereum and Hyperledger Burrow blockchain technologies and compare their different architectures. Second goal is to introduce Caterpillar and find out if it is practical to add Hyperledger Burrow to the project. The hypothesis of this thesis is that Hyperledger Burrow provides more convenient blockchain-based smart contracts and it is practical to implement Burrow to Caterpillar.

2. Background

This paragraph will explain and define related terminology, concepts and technology that are used in the thesis. It will explain the idea behind blockchain and distributed ledger, what are smart contracts and how they can be used for decentralised applications. It will briefly compare private and public blockchain and explain what is Caterpillar.

2.1 Blockchain

Blockchain is a technology that can be programmed to record not just financial transactions but virtually everything of value as it is incorruptible digital ledger of economic transactions [4]. Blockchain is a time-stamped series of immutable records of data, that are stored in an immutable database. It is a distributed, decentralised, public ledger. Its data is managed by clusters of computers called nodes that are not owned by one single entity. Blockchain is called decentralised ledger because the information in it is open for anyone to see. The ledger is a bundle of blocks (changes or transactions) that are chained together. It is mathematically impossible to modify existing ledgers, because blockchain uses cryptographic algorithms.

Blockchain is back-linked record of blocks (transactions), which are timestamped, immutable and in strict order (see figure 1). Each block is distinguishable by a hash, created by utilizing the SHA256 cryptographic algorithm, which are digital fingerprints, that make the blockchain immutable. A block consists of metadata in the header and a lengthy record of transactions that grow in time. As each block is linked to its former parent, the hash is calculated by using all previous transactions and hash or a parent block.

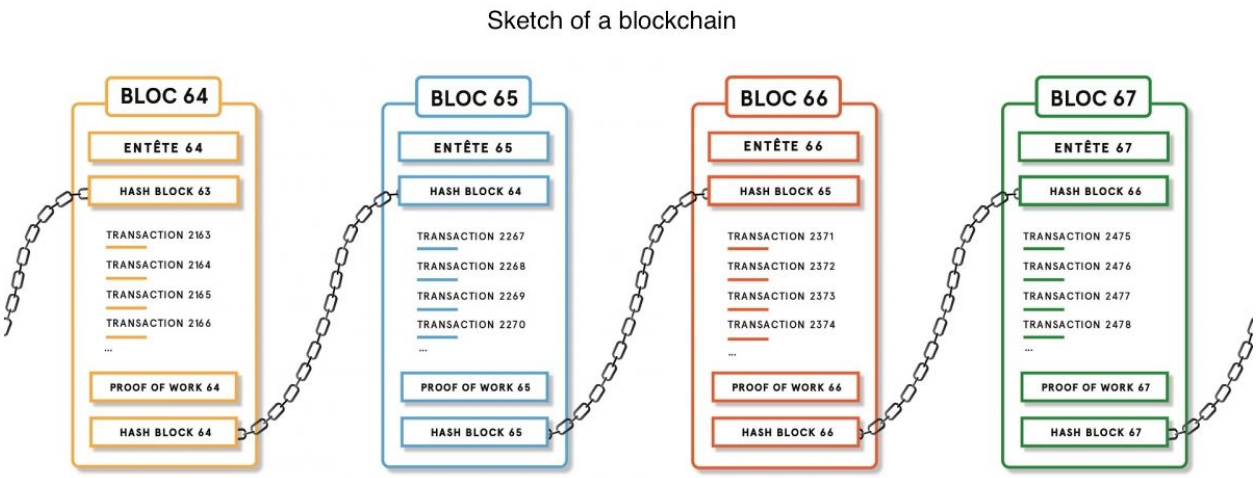

Figure 1. Blockchain structure [5]

2.2 Distributed Ledger

Ledgers [6] are as ancient as money or writing as they are the foundation of accounting. Since the ancient times, ledgers have been the key of economic transactions. They record all the transactions, asset buy-sell deals, contracts and all kinds of payments. First there were clay tablets and papyrus which recorded all important deals which eventually were taken over by the invention of paper. Since the invention of the first mechanical computer in 1822 by Charles Babbage [7], most of ledger recordings have moved to digital form providing easy recordings of data and ledger maintenance with great convenience and speed. Breakthroughs in cryptography and computing power have allowed the creation of distributed ledger.

A distributed ledger [6] is a list of data that is stored, maintained and updated by a list of independent nodes (or participants) in a large network. There are no central authorities that send records to various nodes, but are instead independently constructed by each node. Every node in a network holds a copy of the transactions, coming to its own conclusions and making votes based on the computational solutions to make sure that the majority agrees with the final conclusion. Once consensus has been agreed on over the network, the distributed ledger will be updated.

This kind of system eliminates the possibility of data manipulation. All information stored on the ledger is securely held by cryptography and can only be accessed using keys and cryptographic signatures [8]. Data saved on the ledger will become immutable. Decentralised ledgers are far more secure against cyber-attacks as the attackers need to target every single node in the network.

2.2 Smart contract

Smart contracts, also known as cryptocontracts, were first introduced in 1994 by Nick Szabo, a computer scientist who invented virtual currency "Bit Gold" in 1998, 10 years before Bitcoin was introduced. Smart contracts are lines of codes which are self-executing containing information about terms of an agreement between parties involved in the contract. A decentralized, distributed blockchain network contains the code and the agreements. Smart contract executes itself to produce the output, upon execution, if sets of predefined rules are met. The code allows decentralised automation by verifying and enforcing the condition in an agreement. Agreements and trusted transactions are being carried out among anonymous parties without any central authority or legal system. This allows smart contract to exchange property, money, shares etc, in a transparent manner keeping the system conflict-free. Smart contracts are irreversible, transparent and traceable. [9]

One useful example [10] of using smart contracts is when you need to get a court-registered document as a proof, you need to contact a notary or a lawyer first, then give them the money for the process in return of their service and you would have to wait until the documentation is created and handed back to you. With a smart contract, you would simply get the documentation by paying just for that and it will be done without any third party involvement, such as notary or lawyer in this case. Smart contracts make the rules of agreement and also are responsible for the automatic execution of those rules and obligations.

2.3 Decentralised application (dApp)

A decentralised application is a program that runs on a peer-to-peer network of computers rather than a single computer or a server. A peer-to-peer network is created when two or more computers are connected and share resources without going through a separate server. dApp's are blockchain-enabled websites where smart contracts are used instead of traditional API connections. When traditional websites use API calls for data transfer and databases for data storage, then dApps use smart contracts for data transfer and a blockchain for storing data. When centralized applications use centralized servers to run backend code then dApps have their code running on peer-to-peer network [11].

2.4 Public blockchain vs private blockchain

There are two types of blockchain: private blockchain and public blockchain. When comparing private or public blockchain then we are talking about who can write data onto the blockchain or onto the ledger.

Anyone can join public blockchain network by reading, writing or just participating in the blockchain. They are decentralized meaning that no one has control over them. The network is secure in that the data cannot be changed once its hash been validated [12]. Examples of public blockchain are Bitcoin and Ethereum which are called permissionless blockchain platforms.

Private blockchain is a permissioned blockchain, meaning there can be restrictions on who is allowed to read, write and participate in the network [12]. Examples of permissioned blockchains are Codra, Hyperledger Fabric and Hyperledger Burrow. Ethereum can also be used for building applications with permissioned or controlled access to data, but Ethereum does not provide built-in tools that you can find on a private or permissioned blockchain platform like Burrow.

2.5 Caterpillar

As participants of a deal or just ordinary businesses do not necessarily trust each other, collaborative business processes between mutually untrusting parties can be solved with business process management systems, such as those based on the standard Business Process Model and Notation (BPMN). [13]

Caterpillar is a blockchain-based Business Process Model and Notation (BPMN) execution engine supporting the creation of instances of a process model and allowing users to monitor the state of process instances. Caterpillar uses Ethereum blockchain to maintain each state of a process instance and BPMN-to-Solidity compiler is used to generate smart contracts that are used for workflow routings. This compiler generates smart contracts that can be passed to Solidity compiler which produces EVM bytecode and ABI definitions that are used for deploying the smart contracts to Ethereum. This mean that Caterpillars is built in part to the specification of the Ethereum Virtual Machine [13].

3. Related work

Different blockchain technologies have gained a lot of attention over the past few years. Juniper's Blockchain Enterprise Survey: Deployments, Benefits & Attitudes [14] found out that 65% of large enterprises, these that have over 10 000 employees, are considering or are actively engaged in blockchain deployment [15]. As the cryptocurrencies attracted a lot of attention during the 2017-2018 Bitcoin bubble, a lot of newcomers started to look into different blockchain projects and as a result several blockchain theoretical and practical comparison research papers where written. There is a large number of comparisons between Ethereum and some other blockchain technologies. Research papers have been written about comparisons between Ethereum, Hyperledger Fabric and Hyperledger Sawtooth [16]. As Hyperledger Fabric and Sawtooth are older Hyperledger projects and Burrow is still in incubation stage, there are not many research papers about Hyperledger Burrow as it has not gained so much attention yet. There are few theoretical research papers that mention Hyperledger Burrow briefly, but none explain it in depth [17]. It gives a brief analysis of Hyperledger Burrow history, its purpose and main components. There are more general Ethereum and Hyperledger comparisons, which do not directly talk about Burrow or any other Hyperledger project [18]. There are not any comparison papers which would compare only Ethereum and Hyperledger Burrow. This report underlines the differences between these two platforms, describes what are the main differences from the good and bad side.

The thesis written by Veskus Karl in 2018 compares Ethereum and Fabric. The study compares two given blockchain technologies. During the work, a comparison pet shop application is created for both platforms [19]. This research is most similar the given thesis as its first comparison technology (Ethereum) is the same and the second technology (Hyperledger Fabric) is related to Hyperledger Burrow. Also, this thesis and Karl Veskus thesis both use Pete's Pet Shop as an example, which is created by Truffle Suits framework. The reason why this thesis is using the same example is explained in the following chapters.

This thesis differs from previous works in a way that it compares Ethereum with Hyperledger Burrow technology. As there are no comparison reports or open source example projects written in Hyperledger Burrow, it will be a challenge. Similarly to Sven Mitt master's thesis [20], where he creates a case study about Hyperledger Fabric, I will be comparing two technologies theoretically and practically. I will create two different applications using Ethereum blockchain, Hyperledger Burrow blockchain and smart contracts.

4. Use case

This chapter describes the use case selected for this comparison. It is going to give an overview of Ethereum and Hyperledger Burrow architecture and explain why these were selected for this given thesis comparison.

4.1 Pete's Pet Shop Use Case

Pete's Pet Shop is a use case created by the Truffle framework. Truffle is a developing environment and asset pipeline for blockchain using Ethereum. Truffle was selected because it is one of the most well-known frameworks used by Ethereum users.

Pete's Pet Shop is a website for animal adoptions. It is a simple example of how Ethereum blockchain can be used for these specific transactions with 0 Ether. This website allows adopters to click on the "adoption" button and then a one-way transaction is sent which triggers the smart contract (see figure 3). All the transactions are saved to the blockchain and can be viewed later. Given examples cover transactions, smart contracts, usage of Ether, accounts and implementation with the dApp.

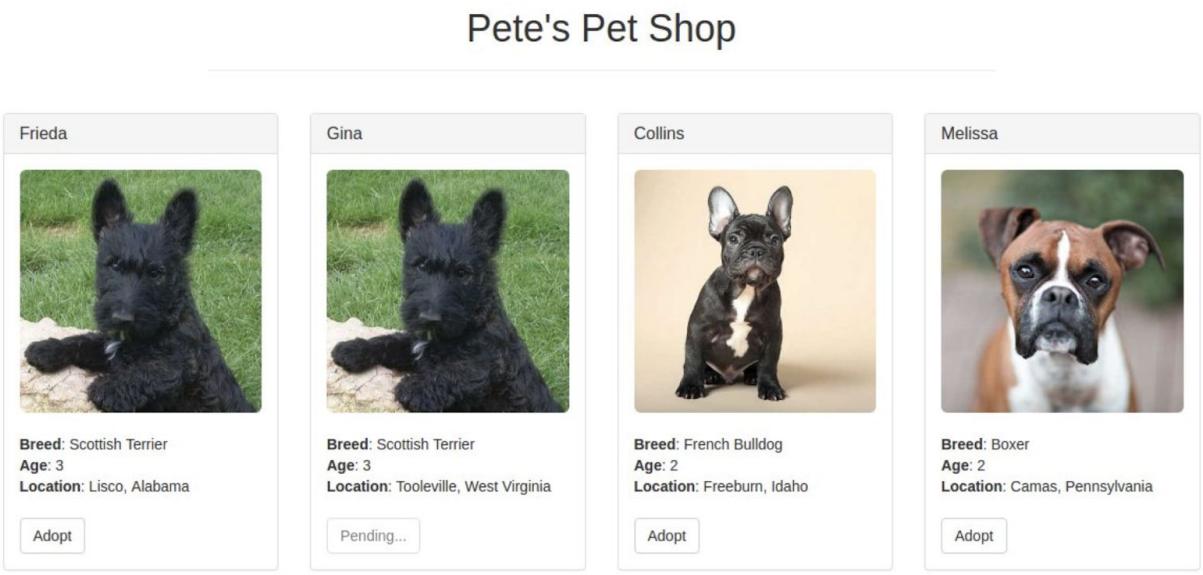

Figure 3. Pete's Pet Shop browser UI view

4.2 Selected Technologies

4.2.1 Ethereum

Ethereum was first introduced at the North American Bitcoin Conference in early 2014 by Vitalik Buterin. Launched in 2015, Ethereum is now one of the most well known and most well-established, open-ended decentralised blockchain software platforms, that enables dApps and smart contracts to be built and run without fraud, control, downtime or interference from third parties. Ethereum has its own programming language called Solidity which is Turing complete.

A turing complete programming language [21] is theoretically capable of expressing all tasks accomplishable by computers; nearly all programming languages are Turing complete if the limitations of finite memory are ignored.

Based on blockchain technology, Ethereum [22] is an open software platform that enabled developers to create and deploy decentralised applications. Ethereum and Bitcoin are similar as they both have distributed public blockchain networks. There are several differences between these two technologies, but the most significant one is that Bitcoin and Ethereum differ substantially in capability and purpose. Bitcoin is mostly focused on peer-to-peer electronic cash system with tracking ownership of digital currency, while the Ethereum blockchain focuses on running smart contracts.

4.2.1.1 Architecture

One of the most important parts of Ethereum is its core invention which is the Ethereum Virtual Machine (EVM) (see figure 4). It is a Turing complete software that is running on Ethereum network. EVM enables to run any program on the network, regardless of the programming language given enough time and memory. Ethereum makes creating blockchain application far more simpler than having to build an entirely original blockchain for each application, Ethereum enables to run all the different decentralised applications or dApps on the same network, the Ethereum network. Ethereum network is a multifunctional network which provides peer-to-peer digital cash payments, creation of decentralised applications, decentralised autonomous organizations, developing and launching other cryptocurrencies etc. [23]

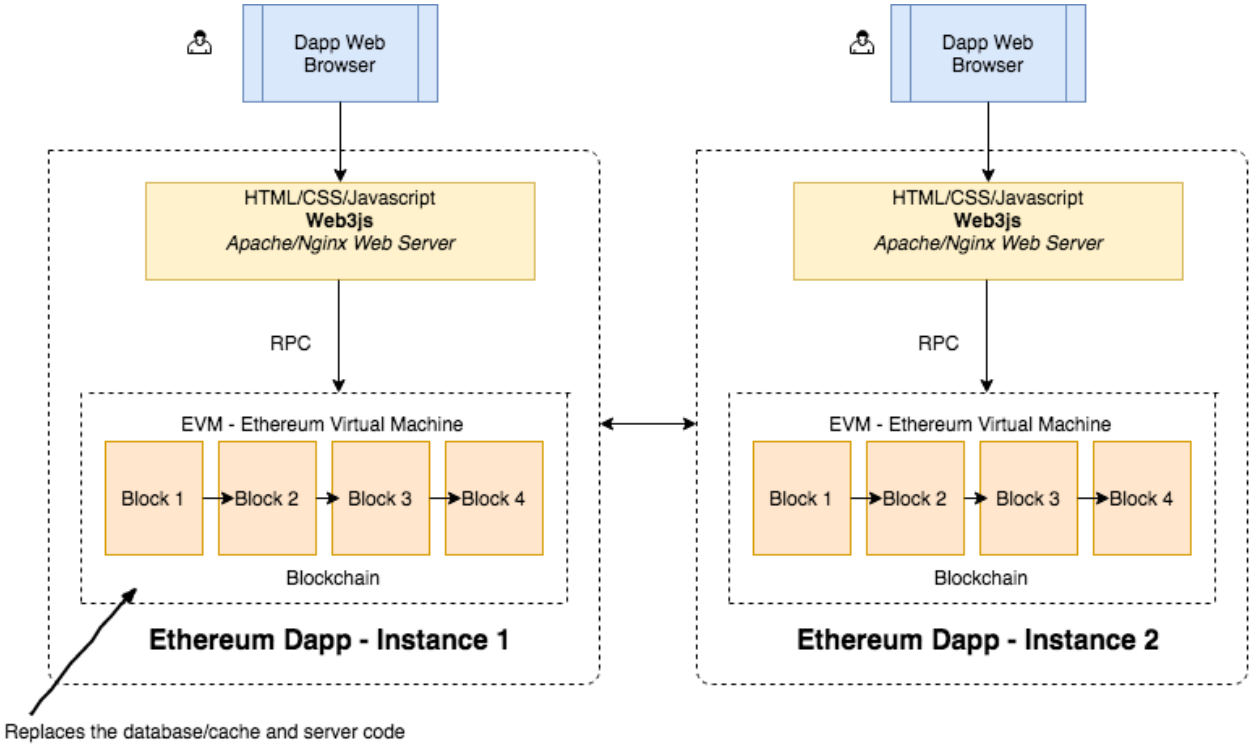

Figure 4. Ethereum Architecture [24]

Ethereum uses a digital currency called Ether (ETH) to power its network. Besides cryptocurrency trading, Ether is fuel for application development as it is used to pay for transaction fees in the network. When you send an ETH or ERC20 token, or do any other type of smart contract related transaction on the blockchain, the used computational power must be paid. These calculations are measured is gas. Regardless of whether transaction succeeds or fails, a payment is required for the computation power. Total gas for the transaction is calculated with this simple formula: gas limit * gas price. [23]

Gas limit is the maximum amount of units of gas you are willing to spend on a transaction. Such limit eliminates the threat of overspending when transactional errors happen. Units of gas necessary for the transaction are already fixed as they are calculated by how much code is executed on the blockchain by the transaction. [23]

Gas price is the price the user is willing to pay for the transaction. Gas price is not fixed and every user can decide on their own what they are willing to give. The higher the gas price user sets, the sooner transaction gets processed in the blockchain, because the miners sort the transaction in a descending order based on the gas price. [23]

4.2.1.2 Accounts

Ethereum blockchain uses public and private keys for each account. The public key is seen by everyone, like an email address and private key must be kept secret as it represents the password for account funds. There are two types of account in the Ethereum ecosystem. First one is the externally owned accounts (EOA) which are a combination of public address and private key. These accounts can be used for sending and receiving Ether to/from another account or sending transactions to smart contracts. The second type of account is contract accounts which don't have a corresponding private key. These accounts are generated once a smart contract is deployed to the blockchain. These accounts can send and receive Ether, they have code associated with them unlike EOA and transactions can only be triggered by an EOA or another smart contract. [25]

4.2.1.3 Wallets

Ethereum wallets are plugins/libraries that help you to store and manage Ethereum accounts. There are nondeterministic and deterministic wallets. Non-deterministic wallets use a given private key to generate a public key from it. There is no limit on how much private keys can be generated, but here is a relation between each pair of keys. Deterministic wallets use seeds, serialized phrases that are human readable, which are used to derive keys from a single starting point. Seeds allow the creation of public addresses without the knowledge of the private key allowing user to easily backup and restore their wallet. [26]

4.2.1.4 Proof of Work and Proof of Stake

Ethereum network is using a efficient record-keeping systems, which is generally called mining. This is a network of miners who are verifying transactions and adding them to the public ledger. As a comparison, banks are usually in charge of keeping accurate records of transactions. They have to ensure that money is not created out of thin air, and that no one cheats and spends money more than they have. This kind of systems that banks use, are not safe against cyber attacks. Mining is is one of the possible solutions that makes decentralized record-keeping possible. [27]

Proof of work (PoW) is a computational challenge or puzzle-solving method, which is used in the Ethereum network for transaction validation. These validators are called miner who run block's unique header metadata through a hash function, only changing the 'nonce value', which impacts the resulting hash value. The main goal of miners is to find a hash that matches the current target. If the miner finds the correct answer it will be awarded with ether and then the block is broadcasted across the network for each

node to validate and add to their own copy of the ledger. It usually takes 12 seconds for some miner to find the correct answer to the puzzle. The profitability depends on miners' the amount of computing power they have and luck. Issue with PoW is that it requires a great deal of computing power to run different calculations to unlock the challenge. Computing translates into high amounts of electricity and power needed for the proof of work. [27]

Proof of Stake (PoS) is another algorithm where nodes can mine or validate block transaction according to how many coins they hold. PoS seeks to address the computing power and the cost of electricity of PoW issued by attributing mining power to the proportion of the coins held by a miner. Developers on Ethereum are planning to switch from PoW to PoS algorithm.

4.2.1.5 Byte Code

Smart contracts are written in a high-level programming language such as Solidity. When sending the smart contract to the blockchain it is compiled to EVM bytecode. Similarly programming language Java is converted to JVM Bytecode before running. This kind of conversion allows developers to use other programming languages to implement smart contracts like Vyper, Pyramid Scheme, Flint, LLL, HAssembly-evm etc. [28]

4.2.1.5 Truffle

There are several JavaScript libraries that are used to interact with the Ethereum blockchain. Just like Python or Ruby on Rails have frameworks to web application development Ethereum network has Truffle and Embark which are the most popular frameworks used by the Ethereum network developers. They abstract away a lot of the complexities of compiling and deploying contracts to the blockchain by providing built in tools and providing a testing framework that can be used to test contracts.

4.2.2 Hyperledger Burrow

Hyperledger Burrow was first introduced by Monax and Intel in December 2014 when it was accepted into Hyperledger Incubator. Hosted by the Linux Foundation it is now one of the Hyperledger projects. It is the first on its kind as is provides modular blockchain client with a permissioned smart contract interpreter. Burrow is built in part to the specification of the Ethereum Virtual Machine (EVM). At the moment of writing this thesis, Burrow is still considered to be in the incubation stage. [29]

4.2.2.1 Architecture

Burrow is constructed out of three main components: the permission Ethereum virtual machine (EVM), the consensus engine and remote procedure call[4] gateway. It acts as a permissioned smart contract application engine whose primary job is executing and processing smart contract programs in an efficient and secure way. Burrow is built the way that it supports application-specific optimization for multi-chain environments. Better known Hyperledger systems are Hyperledger Sawtooth and Hyperledger Fabric. These two are both designed to be highly extensible platforms with a broad range of smart contract operational systems, turning capacity for their runtime and networks. Hyperledger Burrow differs from these two by being focused explicitly on running EVM style smart contracts in a permissioned environment [30]. More specifically Hyperledger Burrow consists of consensus engine, application blockchain interface, smart contract application engine, application binary interface and API gateway (provided by Monax) and permissioned Ethereum virtual machine (see figure 5).

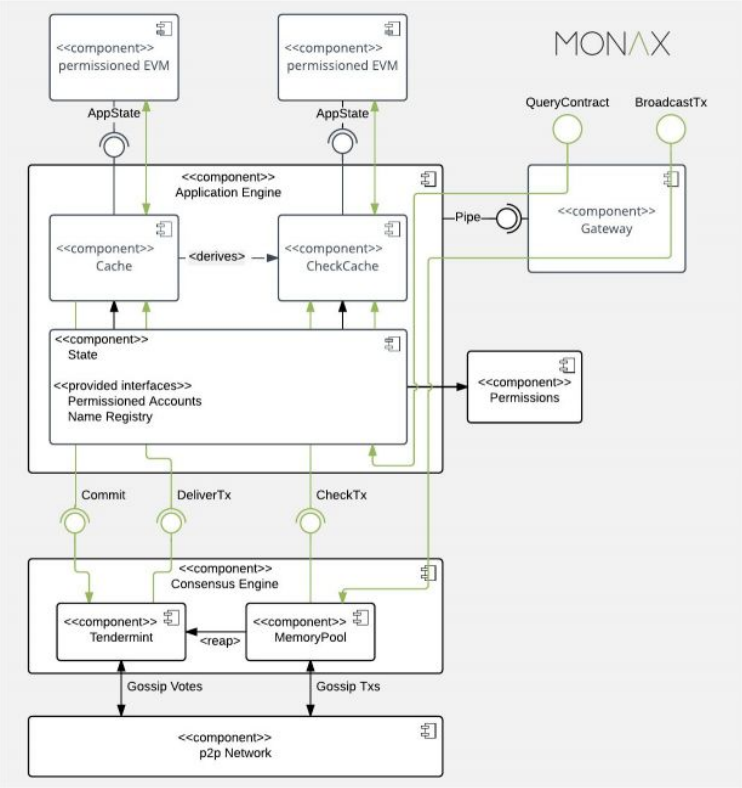

Figure 5. Burrow architecture. [32]

[4] Remote prodcedure call or RPC is an interprocess communication technique that is used for server-client based applications. It can also be known as subroutine call or a funnction call. [31]

Consensus engine is a set of transactions that are ordered and finalised with the Byzantine fault-tolerant Tederminit protocol. Tendermint protocol provides high transaction performance by providing a set of validators and preventing the block chain from dividing into two branches. [17]

Application blockchain interface (ABCI) is an interface that defines the boundary between the application, and the blockchain. This interface enables any programming language to be processed. ABCI allows the consensus engine not to be dependent of the implementation of the contract. [33]

Smart contract application engine makes integration easier for complex business logic. Validated transactions are added to the status of the application in the order as the consensus engine finalized them. If a transaction takes place that invokes the smart contract code, then on a given account will trigger the execution of that account's code in permissioned virtual machine. [17]

Permissioned Ethereum virtual machine is built to Ethereum's operating code specifications and it ensures the appropriate permissions are generated. [17]

Application binary interface (ABI[5]) is an interface that formulates transaction into a binary format that can be processed by the blockchain node. Current Burrow tools provide the functionality to implement, compile and link solidity smart contracts and formulate transactions to call smart contracts on the chain. [17]

API gateway is a system integration and user interface. REST and JSON enable terminals for customers to interact with the blockchain network. Websocket or two way communication channel provides user to subscribe to events, which is especially valuable as the smart contract application and consensus engine deliver unambiguously finalized results of transactions within one blocktime of approximately one second. [17]

[5] An application binary interface (ABI) is in computer science an interface between two binary program moduls. Usually one of them is a library or operating system facility and the oder is program that is begin runned by a user. [34]

5. Implementation

This chapter is about use case implementation made using Ethereum and Hyperledger Burrow. This chapter explains why pet shop use case was created, what tools were used for it and how to replicate the results. It covers the tools what were used for the development and will explain the whole process step by step.

5.1 Ethereum dApp

Ethereum can be built directly on top of Ethereum blockchain, but it is not convenient anymore as several frameworks have been created by the active Ethereum community members. The given Ethereum application is using one of the most popular Ethereum frameworks called Truffle. It is following one of Truffles tutorials called Pete's Pet Shop and it can be found from Truffles homepage[6]. Truffle has provided unfinished code for this project, but the finished implementation with slight modifications can be found from a public repository GitHub.

The given example was written and tested on macOS High Sierra version 10.13.6. Visual Studio Code was used as the integrated development environment (IDE) with Solidity and Javascript plugins. For the purpose of testing and simulation Ganache[7] was used which provides blockchain for Ethereum development in your personal machine. Ganache can be used to develop applications, deploy contracts, or run tests in a local machine.

To start off, we will need to clone the project from GitHub and download all necessary requirements like Git, Nodejs and Ganache. All the instruction can be found from the given Ethereum Pet Shop GitHub repository[8].

5.1.1 dApp smart contracts

The given example uses smart contract to communicate with the ledger. There are several smart contract languages that Ethereum blockchain supports for example: Vyper, Pyramid Scheme, Flint, LLL,

[6] https://truffleframework.com/
[7] https://github.com/trufflesuite/ganache-cli
[8] https://github.com/Villem-OskarOssip/ethereum-petshop

HAssembly-evm, Solidity etc. Our application is using solidity smart contract as is the most widely used, well documented and as this given Truffle already uses this smart contract programming language.

A smart contract [35] is created called *Adoption.sol* in the *contract/* directory. The first line of the code (see figure 6) defines the minimum version of Solidity required. The *pragma* command means that "additional information that only the compiler cares about", while the caret symbol (^) means "the version indicated or higher". The third line of code declares the beginning of the contract class.

```
1   pragma solidity ^0.5.0;
2
3   contract Adoption {
4
5       address[16] public adopters;
6
7       // Adopting a pet
8       function adopt(uint petId) public returns (uint) {
9           require(petId >= 0 && petId <= 15);
10          adopters[petId] = msg.sender;
11          return petId;
12      }
13
14      // Retrieving the adopters
15      function getAdopters() public view returns (address[16] memory) {
16          return adopters;
17      }
18
19  }
```

Figure 6. Adoption smart contract

Solidity is a statically-typed language, meaning data types like string, integers and arrays must be defined. Solidity has a unique type called an address. These addresses are stored as 20 byte values. An address can belong to an account or to a smart contract. One the fifth line of the code we declare *adopters* variable which is an array. Arrays contain one type and can have fixed or variable length size. In our case the variable has length of 16. *Adopters* variable is declared public which gives it automatic getter methods. We use this variable to manage pet adoptions by saving the transaction senders chosen pet ID into the corresponding clients adopters list.

Between lines eight and twelve we declare a function called adopt that takes in pet ID, saves is to the adopters list if given conditions are met and returns the pets ID. In Solidity, the types of both the functions output and parameters must be specified. In our case we will be taking in an integer (pet ID) and returning it in the end. As our example has 16 different pets to adopt we are checking on line 9 that given id has an existing match. If it is successful then *msg.sender* indicates the address of the smart contract or person who called this function. [35]

Between lines fifteen and seventeen we have a function that returns the entire array of adopters. It is used by the UI to update all pet adoption statuses. The function does not take in any parameter but returns a list as *memory* which gives the data location for the variable. The *view* means that the function will not modify the state of the contract. [35]

5.1.2 Deployment

In the terminal, we execute command *truffle compile, which* compiles Solidity to bytecode for EVM [35]. Next we create a new migration script 2_deploy_contracts.js in *migrations/* directory, which is used to migrate compiled contract to the blockchain (see figure 7).

```
1   var Adoption = artifacts.require("Adoption");
2
3   module.exports = function(deployer) {
4       deployer.deploy(Adoption);
5   };
```

Figure 7. Smart contract migration

Truffle defines a migration [35] as a deployment script meant to move application smart contracts from one state to another. Before we can rune migration we need to have blockchain running in our local machine. For this, we are using Ganache and by default running our blockchain on port 7545. Now we can run *truffle migrate* in our terminal to migrate contract to the blockchain.

5.1.3 Front-end

Now that we have contract migrated to the blockchain we need an interface to interact with it. As with Truffle provided Pet's pet shop already provides us with ready built user interface we just need to connect it with the blockchain and contracts. From the *src/* directory, we can find *index.html* file which has the basic page structure and template for displaying the pet shop information (see figure 8). In line 49 we can

see that the file is importing *app.js* which contains code to link front-end with the back-end buy making the button *Adopt* interactable.

```
<div id="petTemplate" style="display: none;">
  <div class="col-sm-6 col-md-4 col-lg-3">
    <div class="panel panel-default panel-pet">
      <div class="panel-heading">
        <h3 class="panel-title">Scrappy</h3>
      </div>
      <div class="panel-body">
        <img alt="140x140" data-src="holder.js/140x140" class="img-rounded img-center"
        style="width: 100%;" src="https://animalso.com/wp-content/uploads/2017/01/Golden-Retriever_6.jpg"
        data-holder-rendered="true">
        <br/><br/>
        <strong>Breed</strong>: <span class="pet-breed">Golden Retriever</span><br/>
        <strong>Age</strong>: <span class="pet-age">3</span><br/>
        <strong>Location</strong>: <span class="pet-location">Warren, MI</span><br/><br/>
        <button class="btn btn-default btn-adopt" type="button" data-id="0">Adopt</button>
      </div>
    </div>
  </div>
</div>
<!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
<!-- Include all compiled plugins (below), or include individual files as needed -->
<script src="js/bootstrap.min.js"></script>
<script src="js/web3.min.js"></script>
<script src="js/truffle-contract.js"></script>
<script src="js/app.js"></script>
```

Figure 8. HTML structure

It starts off by calling *App.init* function (see figure 9). This function is called every time the page is reloaded [35].

```
$(function() {
  $(window).load(function() {
    App.init();
  });
});
```

Figure 9. Page reload function

First, the App.js initializes two global variables and defines the init function. Init function is executed automatically and it creates pet objects for the index.html template and returns App.initWeb3() function call (see figure 10).

```
2     web3Provider: null,
3     contracts: {},
4
5     init: async function() {
6       // Load pets.
7       $.getJSON('../pets.json', function(data) {
8         var petsRow = $('#petsRow');
9         var petTemplate = $('#petTemplate');
10
11        for (i = 0; i < data.length; i ++) {
12          petTemplate.find('.panel-title').text(data[i].name);
13          petTemplate.find('img').attr('src', data[i].picture);
14          petTemplate.find('.pet-breed').text(data[i].breed);
15          petTemplate.find('.pet-age').text(data[i].age);
16          petTemplate.find('.pet-location').text(data[i].location);
17          petTemplate.find('.btn-adopt').attr('data-id', data[i].id);
18
19          petsRow.append(petTemplate.html());
20        }
21      });
22
23      return await App.initWeb3();
24    },
```

Figure 10. Init function

Web3.js [36] is a collection of libraries which allow the interaction with a local or remote Ethereum nodes. The initWeb3 function is for retrieving user accounts, sending transactions and interacting with smart contracts [35] (see figure 11). Between lines 28 and 37 we check if we are using the dApp browser or *MetaMask* where an Ethereum provider is injected into the window object. If yes, then we create web3 object, but we also need to request access to the account on line 32 with *ethereum.enable()*. If there is no Ethereum object then we check for an injected web3 instance. If so, we get its provider and use it to create the object [35]. We reach content between lines 43 and 45 when there are no web3 instances, so we create our web3 object based on local provided (Ganache in this case).

```
26    initWeb3: async function() {
27  // Modern dapp browsers...
28      if (window.ethereum) {
29        App.web3Provider = window.ethereum;
30        try {
31          // Request account access
32          await window.ethereum.enable();
33        } catch (error) {
34          // User denied account access...
35          console.error("User denied account access")
36        }
37      }
38      // Legacy dapp browsers...
39      else if (window.web3) {
40        App.web3Provider = window.web3.currentProvider;
41      }
42      // If no injected web3 instance is detected, fall back to Ganache
43      else {
44        App.web3Provider = new Web3.providers.HttpProvider('http://localhost:7545');
45      }
46      web3 = new Web3(App.web3Provider);
47
48      return App.initContract();
49    },
```

Figure 11. initWeb3

As we now can interact with Ethereum, we need to instantiate smart contracts so web3 knows where to find them and how they work [35]. For that Truffle has provided a library called *TruffleContract* that keeps the information about the contracts in sync with migrations, so that it is not required to change the contracts deployed address manually (see figure 12).

```
51    initContract: function() {
52      $.getJSON('Adoption.json', function(data) {
53        // Get the necessary contract artifact file and instantiate it with truffle-contract
54        var AdoptionArtifact = data;
55        App.contracts.Adoption = TruffleContract(AdoptionArtifact);
56
57        // Set the provider for our contract
58        App.contracts.Adoption.setProvider(App.web3Provider);
59
60        // Use our contract to retrieve and mark the adopted pets
61        return App.markAdopted();
62      });
63
64      return App.bindEvents();
65    },
```

Figure 12. initContract

First, we retrieve the artifact for our smart contract with information about our contract [35]. Then we pass the information into *TruffleContract()*, which creates an instance of the contract and sets web3 provider using the App.web3Provider. Finally we call *markAdopted()* function to update the user interface.

Still in the */src/js/app.js* file, we use this function to update the user interface. First, we access the deploy contract, then call *getAdopters()* on that instance. We declare variable adoption Instance outside of the smart contract and the use *call()* to read data from the blockchain, without having to send a full transaction. After calling *getAdopters()* we loop through all of the adopters. [35] Once a petID with a matching address is found, we disable pets that have already been taken and change the button to "Success" (Figure 13). If an address is not set, it will be equal to empty address.

```
71  markAdopted: function(adopters, account) {
72    var adoptionInstance;
73    App.contracts.Adoption.deployed().then(function(instance) {
74      adoptionInstance = instance;
75      return adoptionInstance.getAdopters.call();
76    }).then(function(adopters) {
77      for (i = 0; i < adopters.length; i++) {
78        if (adopters[i] !== '0x0000000000000000000000000000000000000000') {
79          $('.panel-pet').eq(i).find('button').text('Success').attr('disabled', true);
80        }
81      }
82    }).catch(function(err) {
83      console.log(err.message);
84    });
85  },
```

Figure 13. markAdopted

Between lines 67 to 69, there is a function that handles the button clicking interaction by calling the *App.handleAdopt* function. *HandleAdopt* function uses web3 to get user's accounts and selects the first account as seen on on line 98 (see figure 14). Next we get the deployed contract and send transaction with deploy () which requires *from* address and *petID*. If everything goes smoothly, then the selected pet is adopted. [35]

```
87    handleAdopt: function(event) {
88      event.preventDefault();
89      var petId = parseInt($(event.target).data('id'));
90      var adoptionInstance;
91
92      web3.eth.getAccounts(function(error, accounts) {
93        if (error) {
94          console.log(error);
95        }
96        var account = accounts[0];
97
98        App.contracts.Adoption.deployed().then(function(instance) {
99          adoptionInstance = instance;
100         // Execute adopt as a transaction by sending account
101         return adoptionInstance.adopt(petId, {from: account});
102       }).then(function(result) {
103         return App.markAdopted();
104       }).catch(function(err) {
105         console.log(err.message);
106       });
107     });
108   }
109 };
```

Figure 14. Adoption button handler

5.1.4 MetaMask and interacting with the application

MetaMask is a browser extension for both Chrome and Firefox that lets you interact with dApp's. Using MetaMask is not mandatory, but it is suggested as it gives a better visualisation of the adoption situation. Link how to set up MetaMask can be found from *README* file at Ethereum pet shop GitHub[9]. Given Ethereum pet shop examples works also with out MetaMask extension.

As we have Ganache running and our smart contracts migrated, we will use lite-server[10] library and start the application with *npm run dev* command from the terminal.

5.2 Hyperledger Burrow dApp

As seen from the previous chapter, Ethereum can be built directly on top of Ethereum blockchain, but in many cases it is not convenient. For example, if you have a business process execution engine (BPEE), then you would rather like to hide your transactions from the public eye as the sent data can be sensitive.

[9] https://github.com/Villem-OskarOssip/ethereum-petshop
[10] https://www.npmjs.com/package/lite-server

Given examples is using Hyperledger Burrow to deploy dapp on top of Ethereum blockchain. An example project has been created which is also using Pete's pet shop, but further developing it by deploying it to Burrow. The finished implementation can be found from public repository GitHub[11]. The given example was written and tested on Linux Ubuntu version 16.04. Visual Studio Code was used as the integrated development environment (IDE) with Solidity and Javascript plugins.

5.2.1 Architecture

To start off, we will need to clone the project from GitHub and download all necessary requirements like Git, Node.js and Snak. All the instruction can be found from the given Hyperledger Burrow pet shop GitHub repository. File *account_list.json* contains example 54 accounts, that will be used in this given example. Each account has an address with a public key and private key (see figure 15). But for easier understanding, we have defined a *account.json* which contain one account for our example dapp (these addresses are fakes, generated for the purpose of this example).


```
[
  {
    "address": "6AE5EF855FE4F3771D1B6D6B73E21065ED7670EC",
    "pubKey": "1474C7654BD711B910F48561FCEC85BC5FAE01B1D209CDF6B60D10F141EC7D5B",
    "privKey": "C01E3035C40C2FF009791C36755848F77EA9FAD484E4A38A17355C72A2C5EDB81474C7654BD711B910F48561FCEC85BC5FAE01B1D209CDF6B60D10F141EC7D5B"
  },
  {
    "address": "D7572DA8389D0C3AA64FC8709CA853AFE24F4260",
    "pubKey": "56D2CE5823E4BF1D9621812DE9AFD65DE5786C6096D8C08B4B30C219D8AFC3EF",
    "privKey": "85BB7D2E1856C281190FA174E7478F596BAFF265733C7AE6BE87E0DE10E57F3356D2CE5823E4BF1D9621812DE9AFD65DE5786C6096D8C08B4B30C219D8AFC3EF"
  },
  {
    "address": "B03DD2C47852775208A56FA10A49875ABC507343",
    "pubKey": "DBF0B307F5574A8EA652D6C7CC5C1917330344FB5E5EC296B90101BA477F3602",
    "privKey": "59EF4EC05BFA65581FF4FFCA109B88FB6E63C65E3934040B3E38E8BFD2774E6BDBF0B307F5574A8EA652D6C7CC5C1917330344FB5E5EC296B90101BA477F3602"
  },
```


Figure 15. Account_list.json

As Burrow is providing a modular blockchain client with a permissioned smart contract interpreter, we can use the same smart contracts that were developed for Ethereum pet shop example (Figure 6).

Socket.IO is a JavaScript library that enables real-time, bidirectional and event-based communication. It has a client-side library that runs in the browser and server-side library for Node.js.

In line 9 we use *express.static('src')* to serve static files such as images, CSS and JavaScript files in a directory named *src*. Express is a minimal and flexible Node.js web application framework that provides a

[11] https://github.com/Villem-OskarOssip/hyperledger-burrow-petshop

robust set of features for web and mobile applications. Between lines 10 and 12 we order server to listen on port 3000. When connection is established text is printed to console (see figure 16).

```
9   app.use(express.static('src'));
10  http.listen(3000, function () {
11      console.log('Check out the petstore at http://localhost:3000');
12  });
```

Figure 16. Express and http

Socket.io provides the usage of namespaces for sockets, which essentially mean assigning different endpoints or paths. We use io.on() command on line 16, which we name to define an arbitrary channel, called room, that sockets can join and leave.

Snak is a javascript library for node.js, that makes installing and uninstalling Hyperledger Burrow easy. It has a built-in smart contract compilation, linking and deployment. It provides a interaction with blockchain Burrow directly via Linux terminal.

We use *exec* which is a Node's *child_process* enabling us to access operating system functionalities by running any system command inside child process. By default, pipes for stdout and stderr are established between the parent Node.js process and spawned child. *exec(command, function())* spawns a shell and runs a command within that shell, passing the stdout and stderr to a callback function when complete. We run the snak command which calls out Adoption.sol function getAdopters. If errors do not occur, we will loop over all the adopters and if we find matching address for our example we will save it to a addresses list (see figure 17).

```
14  var addresses = [];
15  var command = '';
16  io.on('connection', function (socket) {
17      command = 'snak call Adoption getAdopters'
18      exec(command,function(err,stdout,stderr){
19          if(!err){
20
21              var array = [];
22              array = JSON.parse(stdout);
23              for (var i = 0; i < array.length; i++ )
24              {
25                  if (array[i].toString() == '8B05651240631E5724CC380D7D249C1FC63F320B')
26                  {
27                      addresses.push(i);
28                  }
29              }
30              console.log(stdout);
31          }
32
33          else{
34              console.log(stderr);
35          }
36      });
```

Figure 17. getAdopters method

We use *socket.on()*, which is an event listener, that can be triggered when clicking the adopt button from the user interface. We use the same kind of logic as with the previous method, but this time we use *snak* to call out *Adoption.sol* method *adopt*. In line 40 we define a command and use *msg.toString()* to pass in selected petID to the *adopt()* function. In line 47 we create another child by calling out *getAdopters* function. We loop over all the adopters and if we find a match we will add it to the addresses list. We use io.emit to send all the addresses to the client so that we can update the UI (see figure 18).

```
39      socket.on('adopt', function (msg) {
40          command = 'snak call Adoption adopt '+ msg.toString();
41          console.log(command);
42          exec(command, async function(err,stdout,stderr){
43              if(err){
44                  console.log(stderr);
45              } else {
46                  command = 'snak call Adoption getAdopters'
47                  exec(command,function(err,stdout,stderr){
48                      if(!err){
49                          var array = [];
50                          array = JSON.parse(stdout);
51                          for (var i = 0; i < array.length; i++ ) {
52                              if (array[i].toString() == '8B05651240631E5724CC380D7D249C1FC63F320B') {
53                                  addresses.push(i);
54                              }
55                          }
56                          console.log(stdout);
57                          io.emit('adopted', addresses);
58                      } else {
59                          console.log(stderr);
60                      }
61                  });
62              }
63          });
64      });
65  });
```

Figure 18. Adopt method

6. Comparison

This section contains comparison of Ethereum and Hyperledger Burrow. The platforms are compared in five aspects: main use case, architecture, consensus algorithm, language and the pet shop dApp implementation.

6.1 Main use case

By main use case, Ethereum and Hyperledger Burrow differ in many ways. Ethereum was initially created because of Bitcoin in that time was mostly used for cryptocurrency transaction and there was no alternative technology that could be used to build decentralised applications. By now there are hundreds of applications built on Ethereum, for example, Augur which is a decentralized market for forecasting real-world events, ETHlend for lending marketplace for cryptocurrencies, Gitcoin for easier way to monetize or interact work in open source software, Akasha for building decentralized social network on Ethereum, a game called CryptoKitties etc. Ethereum is also used for initial Coin Offerings (ICO), which is a way of fundraising. Ethereum also has its own cryptocurrency which is widely traded over the world, listed as the second biggest cryptocurrency with total market value of 17,7 billion (03.05.19) USD.

Hyperledger Burrow, on the other hand, has three primary aims: to be a good compliant and simple EVM library, to be light, fast lean and single-process full Tendermint / EVM permissioned ledger with transaction finality and to provide a practical bae for EVM extensions in a many-chain world. As Burrow is still in incubation phase it has not developed a fan base who would already be building dApps with it. Burrow does not have its own cryptocurrency as it is not in its main purposes list, but still currency could be implemented on top of Burrow if the developers decide to do it as Burrow's technology can support it.

6.2 Architecture

Hyperledger Burrow and Ethereum are designed for a variety of different reasons. Ethereum is designed to be an open-ended decentralised blockchain software platform for any kind of dApps. It offers peer-to-peer digital cash payments, creation of decentralised applications, developing and launching other cryptocurrencies etc. It is designed to be permissionless and transparent. This means that the data is stored and shared all over the network.

Hyperledger Burrow, however, is designed to implement smart contracts in enterprise networks. Its main functionality is to execute smart contracts on an authorized blockchain which is compatible with the Ethereum Virtual Machine. With that, it provides modular and flexible solutions for a private permissioned blockchain by providing security and confidentiality. Burrow gives node customizable access by running multiple chains in parallel. For example [37] when node A and node B use chain 1, while node B and node C use chain 2, so node A has no access to the transactions between B and C.

6.3 Consensus Algorithm

Ethereum is currently using proof-of-work consensus algorithm, but in the mid of 2019 Ethereum network developers are planning to switch to proof-of-stake. Currently, Ethereum if facing problems with PoW because of the high energy cost, increased strain on the environment, associated adverse media coverage, increasing centralization of mining operations, and low transaction throughput.

Burrow has proof-of-stake Tendermint[12] consensus protocol, which is full Byzantine fault tolerant without relying on specialized hardware. PoS algorithm provides for a more scalable blockchain with higher transaction throughput and has almost no energy costs and no bad effects on the environment.

6.4 Transactions

Ethereum blockchain currently supports roughly 15 to 20 transactions per second, because of a hard-coded limit on computation per block. Compared to Visa which supports 1700 transaction per second on average and Bitcoin which supports between 2 and 8 transactions per second (depends on the number and size of transactions). On the other hand, Burrow, which is using Tendermint, can handle transaction volume at the rate of 10,000 transactions per second [38]. All the transactions speeds are compared with 250 byte transactions size.

6.5 Language

Ethereum has its own programming language called Solidity. It is easy to use as it is inspired by scripting programming language Python and JavaScript. As all code that is deployed is converted to JVM Bytecode before running, allows developers to use other programming languages such as Vyper, LLL, Flint, etc. As Hyperledger Burrow is built to follow the Ethereum specification, it supports all the same

[12] Tendermint is software for securely and consistently replicating an application on many machines [37]

programming languages as Ethereum does. This means that there is no need to learn new programming languages if there is a need to use Burrow.

6.6 Pet Shop Use Case

Implementing Pete's pet shop example provided by Truffles, was fairly easy, as its tutorial contained all the necessary instruction. Writing smart contracts and deploying them is simple as there are a lot of articles, instructions, examples projects or even blog posts on the web. Ethereum has a big developer community who have developed various tools and frameworks to make dApp development simpler.

For the author of the given thesis, it was far more difficult to implement the same application on top of Hyperledger Burrow. It was time-consuming to find all necessary tools and to come up with implementation idea, as Burrow is still in incubation stage, there are only a few articles that talk about Burrow's architecture, there are no open source examples or guides on how to implement Burrow. When we compare Ethereum pet shop and Hyperledger Burrow pet shop we can already see from the README file that setting up and running Burrow is more complex. Eventually working example was created, with the exception that it only works on the Linux operating system.

7. Conclusion

The main goal of this thesis was to give an overview of Ethereum and Hyperledger Burrow blockchain technologies and compare their different platforms. For that author provided an overview of the blockchain and the distributed ledger technologies covering main architecture concepts and main use cases.

We managed to find out that Ethereum is easy to use public blockchain. Ethereum Virtual Machine supports many different programming languages that have a lot of libraries to make development easy. There is also a big eco-system that supports the development of Ethereum by providing articles, research papers, example projects, libraries, frameworks or even blog posts. We found out that the main issue with the Ethereum network is that its transaction speed is quite slow when we compare it with other alternatives projects that support decentralised application development. As Ethereums main trademark is to be permissionless and transparent, it eliminates the possibility to have permissioned transaction in its networks. Hyperledger Burrow, in contrast, is designed to implement smart contracts in enterprise networks by providing permissioned smart contracts. For consensus algorithm Burrow is using Tendermint which is offering almost 1000 times faster transaction speed.

For comparison, we implemented same use cases on top of Hyperledger Burrow and Ethereum. During the implementation, we demonstrated how to build decentralized applications on Ethereum and Hyperledger Burrow. We found out that it is fairly easy to build your first dApp on Ethereum as it is a very popular framework, it has a big eco-system with great support of different libraries and frameworks. Hyperledger Burrow, on the other hand, has only a minimum set of tools, as it is still in incubation stage.

In conclusion, given thesis showed that Burrow provides more private blockchain-based smart contracts and it is in part of the specification of EVM, as it supported Ethereum petshop example. This means we have confirmed the first hypothesis of the given thesis which raised the question if Hyperledger Burrow provides more private blockchain-based smart contracting. As Caterpillar is a business process execution engine, which transferred sensitive data between parties, and is built on top of Ethereum Virtual Machine, we can conclude that it is practical to implement Burrow to its architecture, as Burrow supports dApps built on top of EVM and protecting vital transactions with permissioned smart contracts, and with that we have confirmed the second hypothesis of this report.

For future research, the author suggests trying to implement Hyperledger Burrow to more complex applications running on EVM, as the given examples were basic. For example business process execution engine Caterpillar which uses far more complex smart contracts.

8. References

[1] Bitcoin magazine, “Yes, Bitcoin Can Do Smart Contracts and Particl Demonstrates How,” 07.03.2019. [Online]. Available: https://bitcoinmagazine.com/articles/yes-bitcoin-can-do-smart-contracts-and-particl-demonstrates-how/ [Accessed: 07-March-2019].

[2] Brock, E. “How do Ethereum Smart Contracts Work? It's Deceptively Simple,” 12.02.2018. [Online]. Available: https://www.verypossible.com/blog/how-do-ethereum-smart-contracts-work-its-deceptively-simple [Accessed: 07-March-2019].

[3] Hyperledger, “Hyperledger Burrow,” 2019. [Online]. Available: https://www.hyperledger.org/projects/hyperledger-burrow [Accessed: 07-March-2019].

[4] Blockgeeks, “What is Blockchain Technology? A Step-by-Step Guide For Beginners,” 2016. [Online]. Available: https://blockgeeks.com/guides/what-is-blockchain-technology/ [Accessed: 21-March-2019].

[5] L. Pinsar, “Deploy Your First Ethereum Smart Contract on a Blockchain!,” 31.01.2019. [Online] Available: https://blog.theodo.fr/2018/01/deploy-first-ethereum-smart-contract-blockchain/ [Accessed: 07-May-2019].

[6] N. Bauerie, “What is a Distributed Ledger?,” 2019. [Online]. Available: https://www.coindesk.com/information/what-is-a-distributed-ledger [Accessed:22-April-2019].

[7] Computer Hope, “When was the first computer invented?,” 13.11.2018. [Online]. Available: https://www.computerhope.com/issues/ch000984.htm [Accessed:21-April-2019].

[8] C. Majaski, “Distributed Ledger Definition,” 25.01.2019. [Online]. Available: https://www.investopedia.com/terms/d/distributed-ledgers.asp [Accessed: 22-April-2019]

[9] J. Frankenfield, “Smart Contracts”, 14.04.2019. [Online]. Available: https://www.investopedia.com/terms/s/smart-contracts.asp [Accessed:10-April-2019].

[10] M. Pratap, “Everything You Need to Know About Smart Contracts: A Beginner’s Guide”. 29.08.2018. [Online]. Available: https://hackernoon.com/everything-you-need-to-know-about-smart-contracts-a-beginners-guide-c13cc138378a [Accessed:10-April-2019].

[11] BlockchainHub, “Decentralized Applications – dApps,” 2019. [Online]. Available: https://blockchainhub.net/decentralized-applications-dapps/ [Accessed:17-April-2019].

[12] D. Massessi, “Public Vs Private Blockchain In A Nutshell,” 12.12.2018. [Online]. Available: https://medium.com/coinmonks/public-vs-private-blockchain-in-a-nutshell-c9fe284fa39f [Accessed:17-April-2019].

[13] O.L. Pintado, “Caterpillar: A Business Process Execution Engine on the Ethereum Blockchain,” 22.04.2019. [Online]. Available: https://arxiv.org/pdf/1808.03517.pdf [Accessed: 03-May-2019].

[14] Juniper Research, “Blockchain Enterprise Survey: Deployments, Benefits & Attitudes (Second Edition),” 11.09.2018. [Online]. Available: https://www.juniperresearch.com/researchstore/fintech-payments/blockchain-enterprise-survey/deployments-benefits-attitudes [Accessed: 21-March-2019].

[15] M. Huillet, “Blockchain’s Popularity Among Large Enterprises Soared 11% This Year, Survey Finds,” Cointelegraph, 2019. [Online]. Available: https://cointelegraph.com/news/blockchains-popularity-among-large-enterprises-soared-11-this-year-survey-finds [Accessed: 21-March-2019].

[16] Sitoh, P. “What are the differences between Ethereum, Hyperledger Fabric and Hyperledger Sawtooth?”, Medium, 2018. [Online]. Available: https://medium.com/coinmonks/what-are-the-differences-between-ethereum-hyperledger-fabric-and-hyperledger-sawtooth-5d0fc279d862 [Accessed: 21-March-2019].

[17] Medium, “Hyperledger Burrow v0.19.0,” 2018. [Online]. Available: https://medium.com/coinmonks/hyperledger-burrow-v0-19-0-633a143a5047 [Accessed: 21-March-2019].

[18] Edureka, “Hyperledger vs Ethereum – Which Blockchain Platform Will Benefit Your Business?,” 06.12.2018. [Online]. Available: https://www.edureka.co/blog/hyperledger-vs-ethereum/#keydifferences [Accessed: 21-March-2019].

[19] K. Veskus, “Ethereum versus Fabric – võrdlev analüüs,” 2018. [Online]. Available: https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=61989&year=2018 [Accessed: 21-March-2019].

[20] S. Mitt, “Blockchain Application - Case Study on Hyperledger Fabric,” 2019. [Online]. Available: https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=62247&year=2018 [Accessed: 21-March-2019].

[21] Wikipedia, “Turing machine,” 2019. [Online]. Available: https://en.wikipedia.org/wiki/Turing_machine [Accessed: 03-May-2019].

[22] Blockgeeks, “What is Ethereum? Most Comprehensive Guide Ever,” 12.09.2018. [Online]. Available: https://blockgeeks.com/guides/ethereum/ [Accessed:22-April-2019].

[23] B. Prableen, “Bitcoin Vs Ethereum: Driven by Different Purposes,” Investopedia, 15.02.2019. [Online]. Available: https://www.investopedia.com/articles/investing/031416/bitcoin-vs-ethereum-driven-different-purposes.asp [Accessed: 24-April-2019].

[24] Zastrin, “Ethereum Architecture,” 2019. Available: https://www.zastrin.com/courses/ethereum-primer/lessons/1-5 [Accessed: 07-May-2019].

[25] Zastrin, “What is an Ethereum account?,” 2019. [Online]. Available: https://www.zastrin.com/courses/ethereum-primer/lessons/2-4 [Accessed: 24-April-2019].

[26] Zastrin, “What is an Ethereum wallet?,” 2019. [Online]. Available: https://www.zastrin.com/courses/ethereum-primer/lessons/2-5 [Accessed: 24-April-2019].

[27] Hertig. A, “How Ethereum Mining Works,” Coindesk, 2019. [Online]. Available: https://www.coindesk.com/information/ethereum-mining-works [Accessed: 24-April-2019].

[28] Zastrin, “Byte Code,” 2019. [Online]. Available: https://www.zastrin.com/courses/ethereum-primer/lessons/2-7 [Accessed: 24-April-2019].

[29] Hyperledger, “Hyperledger Burrow,” 2019. [Online]. Available: https://www.hyperledger.org/projects/hyperledger-burrow [Accessed:16-April-2019].

[30] G. Motroc, “Blockchain development made easy: Getting started with Hyperledger Burrow,” 22.08.2018. [Online]. Available: https://jaxenter.com/hyperledger-burrow-interview-kuhlman-148519.html [Accessed:16-April-2019].

[31] A. Onsman, “Remote Procedure Call (RPC),” 11.10.2019. [Online]. Available: https://www.tutorialspoint.com/remote-procedure-call-rpc [Accessed:16-April-2019].

[32] Hyperledger, “Hyperledger Burrow (formerly eris-db),” 28.03.2017. Available: https://www.hyperledger.org/wp-content/uploads/2017/06/HIP_Burrowv2.pdf [Accessed: 08-May-2019].

[33] Tendermint ABCI. 06.2018. [Online]. Available: https://github.com/tendermint/abci [Accessed:16-April-2019].

[34] Wikipedia, “Application binary interface,” 14.02.2019. [Online]. Available: https://en.wikipedia.org/wiki/Application_binary_interface [Accessed:16-April-2019].

[35] “Pete’s Pet Shop”, Truffle Suits, 2019. [Online]. Available: https://truffleframework.com/tutorials/pet-shop [Accessed: 25-April-2019].

[36] Web3.js, “web3.js - Ethereum JavaScript API,” 2019. [Online]. Available: https://web3js.readthedocs.io/en/1.0/ [Accessed: 26-April-2019].

[37] P. Suprunov, “5 Hyperledger Projects In Depth”, 21.09.2019. [Online]. Available: https://medium.com/practical-blockchain/5-hyperledger-projects-in-depth-3d14c41f902b [Accessed: 03-May-2019].

[38] Tendermint, “What is Tendermint?,” 2019. [Online]. Available: https://tendermint.com/docs/introduction/what-is-tendermint.html#tendermint-vs-x [Accessed: 03-May-2019].

Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, Villem-Oskar Ossip,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,
   Ethereum blockchain and HyperLedger Burrow blockchain comparative analysis,
   supervised by Orlenys López-Pintado.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Villem-Oskar Ossip
10/05/2019