University of Amsterdam
System & Network Engineering,
MSc

Developing an Ethereum Blockchain Application

Research Project 2

Written by:
Nikolaos Petros Triantafyllidis
Nikolaos.Triantafyllidis@os3.nl

Supervised by:
Oskar van Deventer, TNO
oskar.vandeventer@tno.nl
Erwin Middelesch, TNO
erwin.middelesch@tno.nl

February 19, 2016

Abstract

This project aims to evaluate Ethereum, a decentralized platform for the deployment of Smart Contracts on the Blockchain. Our approach is to initiate initiate a private Ethereum test network, on which we will develop and deploy a decentralized Smart Contract application. The steps towards building the development environment as well as the developed application are demonstrated in this report. Our gathered experiences as well as our estimation about the potential of the Ethereum project are presented in our evaluation.

Contents

1 Introduction 3
- 1.1 Research Question . . . 5
- 1.2 Ethical Considerations . . . 5

2 Background Information 6
- 2.1 Bitcoin . . . 6
- 2.2 Blockchain . . . 8
- 2.3 Proof of Work & Proof of Stake . . . 13
- 2.4 Smart Contracts . . . 14
- 2.5 Cryptographic hash functions . . . 15
- 2.6 Turing Completeness . . . 15

3 The Ethereum Project 17
- 3.1 Ethereum history . . . 20
- 3.2 Ethereum projects . . . 21
- 3.3 Ethereum implementations . . . 22
- 3.4 Ethereum concepts . . . 23
  - 3.4.1 Accounts . . . 23
  - 3.4.2 Contracts and Transactions . . . 24
  - 3.4.3 Registrars . . . 25
  - 3.4.4 Ether . . . 26
  - 3.4.5 Gas . . . 27
  - 3.4.6 Mining . . . 29
- 3.5 Solidity . . . 30
  - 3.5.1 Types . . . 31
  - 3.5.2 Events . . . 31
  - 3.5.3 Functions . . . 32
  - 3.5.4 Function Modifiers . . . 32
- 3.6 Building a test network . . . 33

4 TheDistributed Application 36
- 4.1 The Contract in Natural Language . . . 37
- 4.2 The Justice Token . . . 38
- 4.3 Contract entities . . . 39

4.4 Public Functions 39
4.5 Limitations and Proposed Extensions 42
4.6 Setup scripts 43

5 Evaluation 44
5.1 Difficulty 45
5.2 Computational Power and Storage 46
5.3 Cost 46
5.4 Security 47

6 Conclusion 48

7 Future Work 50

Bibliography 51

Appendix: Source code 57

Chapter 1

Introduction

Since its inception in 2008, Bitcoin [1] has managed to gain a lot of momentum and popularity and is now considered a stable alternative currency and economy system, as well as a strong subculture movement.

The underlying technology of Bitcoin, the blockchain, a distributed database model used to validate the financial transactions in a decentralised manner, is considered tamper proof [2]. For that reason a blockchain based approach can be used for applications beyond monetary, especially in contexts where anonymity, privacy and censorship resistance are important aspects.

A new tool called Ethereum[3] was first described in 2013 and was officially launched in 30 July 2015. Ethereum is a platform designed to run smart contracts over a decentralised network of peers. A smart contract in the concext of Ethereum is described as 'an application that runs exactly as programmed without downtimes, censorship, fraud or third party interference' [3].

The Ethereum project claims that its mission is to fully decentralize the Internet 'as it was supposed to work'[3] by providing a platform on top of which anyone can start a decentralized Internet service, secured by the blockchain. The hypothesis posed is that Ethereum will make it easy to launch blockchain-based applications without needing to start a new blockchain protocol or cryptocurrency.

The aim of this project is to test the validity of this hypothesis by building a decentralized application on top of the Ethereum network. Moreover, gaining insights on the inner workings of Ethereum and development experience on the platform will allow us to empirically evaluate Ethereum in terms of ease of use, development time, performance, added value, and future potential.

The rest of this introductory chapter is dedicated to presenting our research questions as well as any ethical considerations that we came across during the completion of this project. The second chapter provides the definitions as well as additional information regarding fundamental concepts needed for understanding the context of this report. The next chapter gives an in-depth presentation of the Ethereum project and its surrounding concepts. The fourth chapters describes the application built for the needs of the project while the fifth chapter presents our gathered experiences and evaluation of Ethereum. The sixth chapter concludes the report and answers the research questions and the last chapters suggest future extensions of the project.

1.1 Research Question

The main research question we will try to answer in this project is the following: Can Ethereum be directly used to rapidly deploy meaningful and sufficiently performing trusted applications with added value over traditional approaches?

To make the above question more clear we can analyse it to the following subquestions:

1. Can Ethereum be used to deploy non-trivial applications?
2. Can we achieve acceptable application performance even with the currently present constraints?
3. What is the typical time needed for an Ethereum application to be developed and launched?
4. What is the added value of using Ethereum over a more traditional development approach?

1.2 Ethical Considerations

No ethical issues arose during the completion of the project. The Ethereum contracts developed for the needs of this project were executed on a private test network. No connections to the live Ethereum Frontier network were made, and thus no 'real' Ether was spent. No personally identifiable information or any otherwise sensitive data was gathered during this project.

Chapter 2

Background Information

We find it useful at this point to give working definitions as well some background information for certain key concepts whose basic understanding is fundamental in the following chapters. Form a firm grasp of the concepts below, the reader is advised to study the corresponding bibliography.

2.1 Bitcoin

Bitcoin is described as a consensus network that constitutes a system for online payments as well as a digital currency [5]. It is considered the first fully decentralized payment network which is capable of operating on a pure peer-to-peer basis outside the control of any central authority [1].

Transactions in the Bitcoin network are maintained and verified in a distributed transactions ledger called the blockchain [7]. The system is secured against double-spending and reversed transactions through a process called mining, which involves solving certain cryptographic hash puzzles as a Proof of Work. Each transaction is also signed with the private key of the transaction initiator and addressed to the public key of the recipient [2]. Because of the use of these cryptographic features, the term Cryptocurrency has emerged.

Bitcoin was first described in 2008 in a white paper authored by the, still unidentified, person or group of people behind the pseudonym Satoshi Nakamoto [7]. In January 2009 the first Bitcoin client software was released as an open-source project and the Bitcoin network came to life.

This was also the time when the first bitcoin transaction took place. Since then, Bitcoin has seen big growth as more and more companies and organizations started accepting it as a method of payment. The popularity of the currency has been reflected in the, ever fluctuating, Bitcoin to US Dollar exchange rate. Indeed, a single Bitcoin had started to be sold for 0.05$ at its creation days, peaking at 1242$ in 2013, a price comparable to an ounce of gold [4]. The universal symbol for the bitcoin currency is BTC.

A number of factors related to the Bitcoin network security, bugs in the client application software implementations as well as several events in the global economy have either raised or lowered confidence in Bitcoin, and have in turn affected the Bitcoin price, which at the moment (January 2016) fluctuates around the 400$ area. The above issue with the price instability as well as other factors, inherent in the very nature of Bitcoin, has led many, including its strong supporters[6], to question its viability as a stable alternative payment system.

Bitcoin, however, has brought the technology that allows us to run fully decentralized applications and eliminate the trust to any central authority or server infrastructure. Thus, it is essential to understand the inner workings of Bitcoin in order to be able to grasp the concepts that will be presented in the following chapters. These terms will be explained in the sections below.

2.2 Blockchain

The blockchain is the publicly verifiable, distributed transactions ledger that secures the Bitcoin network against double-spending, forgery and reversed transactions.

In an abstract view, the blockchain is a data structure that consists of time ordered, linked blocks that contain a number of transactions. The blocks are linked in the sense that each block includes the ID of the previous block in the chain [2]. The ID of each block itself is the cryptographic hash of its contents. Thus it becomes apparent that each block depends on the previous block thus forming a chain that in order to be recreated all blocks leading to the current one will have to be recreated as well.

To make the function of the blockchain more apparent let us look at the following naive scenario.: User Bob wants to send an amount of Bitcoin (let's assume 1.5 BTC in this example) to user Alice. Bob makes a transaction referencing a number of previous transactions (called 'input' in the Bitcoin protocol) whose value sums to 1.5 BTC, and is addressed to Alice. This transaction is broadcast to the Bitcoin network. Now the network nodes have to confirm that, the referred input does, firstly, indeed belong to Bob and secondly it is unspent, i.e. has not been referenced as the input of a previous transaction. Once this confirmation has been granted the ownership of 1.5 BTC is transferred from Bob to Alice. Now Alice can proceed to provide the service that Bob is paying for.

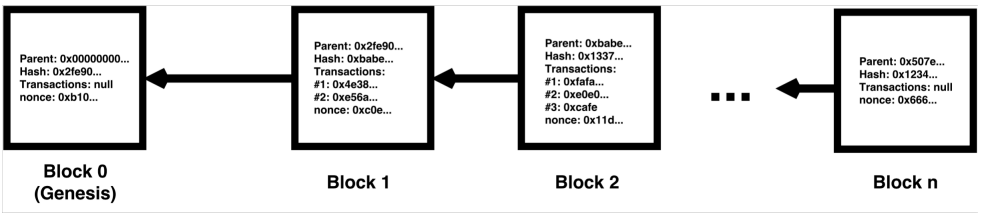

Figure 2.1: Schematic representation of the Blockchain

What if, however Bob has no intention to pay Alice the 1.5 BTC and, in parallel to the first transaction, sends a second transaction sending the same 1.5 BTC back to himself? There is a high chance that certain nodes receive the malicious transaction first and the whole network reaches the consensus that the 1.5 BTC transferred from Bob to himself is valid. In that case Alice has provided the service for free. Thus, there needs to be a mechanism that provides orderign for the transactions. That is where the blockchain is of value.

Once the transactions have reached the nodes and have been verified as valid, certain work towards discovering an appropriate grouping of these valid transactions. This process is called mining and the nodes who participate miners. Each one of these groups is called a block and points to the previous block of grouped transactions, thus forming a chain (hence the name blockchain). Once a block is inserted in the blockchain, the transactions it contains are considered accepted by all nodes in the network.

However, merely forming a group of transactions and suggesting it to the network would mean that anybody would be able to do it which would mean, for one, that the network would be flooded with simultaneous blocks and consensus would never be reached. Secondly it would be possible for anyone to forge blocks with the transactions they would like to include to the network, which would not actually solve the problem of securing the Bitcoin system. For that reason mining is a very demanding process in terms of computational time and resources.

Each block contains a reference to the previous block, a set of new transactions plus a random value (nonce). The contents of the block are hashed and if the output of the hash function is below a predefined target then the block is considered mined and gets broadcast to the network. Thus, mining basically means discovering the right nonce which, together with the contents of the block will produce the desired hash. Due to the properties of cryptographic hash functions (discussed below) it is practically impossible to try and cheat the network by precomputing a set of nonces. The fact that the solution to this cryptographic problem has been discovered by the nodes is considered as Proof of Work and for that reason the block is accepted by the network. In the current implementation of the Bitcoin protocol the miners are rewarded with 25 BTC for the effort they invest into the mining process.

The current difficulty of Bitcoin mining, reflected in the predefined target, means that a new block is discovered in the network approximately every 10 minutes. There is always of course the probability that two blocks are discovered and transmitted at the same time. In this case the blockchain is forked and the nodes continue to mine on the branch they have received first. The consensus is resolved again once a new block arrives and the chain with the longest length, i.e. the chain with the biggest proof of work is considered the valid branch. The transactions in the stale branch are returned to the transaction pool, to be mined at a later stage.

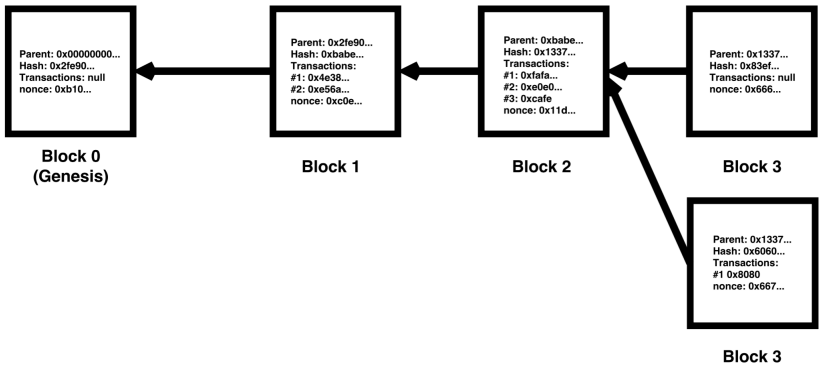

Figure 2.2: Schematic representation of a forked Blockchain.

The above example shows that the blockchain is actually tamper-proof. If a malicious user would like to change a transaction that is deeper within the chain that would mean that they would have to recompute all the blocks that follow the block containing that transaction. That would mean they would have to perform a very expensive operation multiple times in the row, on the same time beating the rest of the mining network in the race of discovering new blocks. This power of the blockchain has led to its use for applications outside Bitcoin.

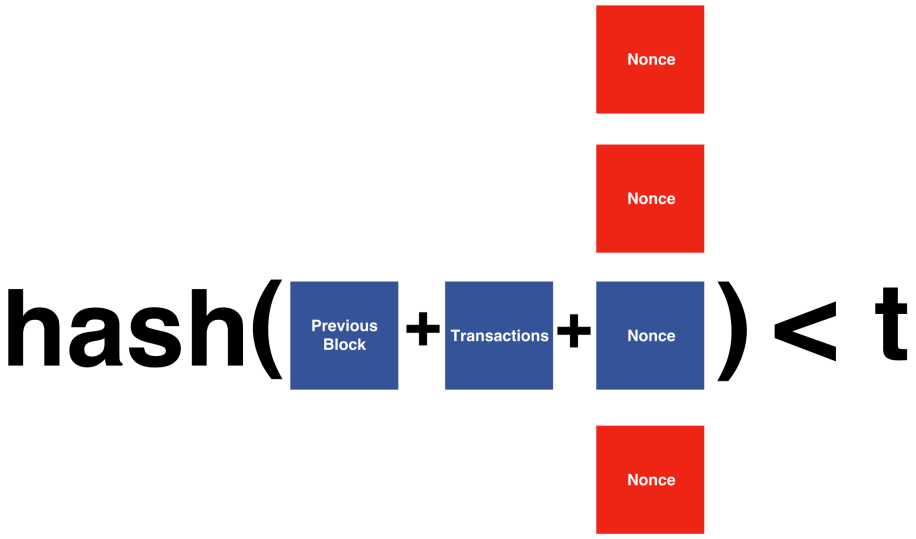

Figure 2.3: Schematic representation of mining. Hashing the contents of the parenthesis should result to a value lower than target t. Mining involves finding the correct nonce that will allow the production of such a value.

2.3 Proof of Work & Proof of Stake

We have mentioned above that the Bitcoin network miners have to solve certain cryptographic puzzles as a Proof of Work. The concept as well as the term Proof of Work, however, has preceded Bitcoin. Proof of Work (POW) in general is a measure used to prevent the abuse of a service, by requiring the service requester to spend a considerable amount of of computational resources and time before the service can be granted [8] [9].

There are two flavors of POW protocols, challenge-response and solution-verification. Challenge-response protocols assume a client-server architecture, where the client requests a service from a server and in turn gets provided with a challenged to solve. Once the solution is submitted and verified the service is granted.

In solution-verification POW protocols the challenge is self produced by the party that needs to prove their work. Once the puzzle gets solved the solution together with the request are sent to the interested remote parties, which in turn verify the solution and grant the requested service. It is such a class of a POW protocol that Bitcoin and other cryptocurrency systems utilize.

A different approach to Proof of Work has been suggested for verifying cryptocurrency transactions, called Proof of Stake (POS). In this approach, instead of having to prove an amount of work spent on the block generation, the nodes have to prove ownership of their currency balance. In POS systems the blocks are mined by the nodes voting on which will be the next block in the chain. Voting rights are distributed according to the 'stake' each node has in the network. For example if a node has 1% of the total currency available they also hold 1% of the voting rights on the blockchain.

The very fact the voting rights are distributed according to stake in the currency share creates problems leading to centralization and monopolization of the blockchain around the nodes with the biggest stake. Moreover, since the creation of blocks if costless, this opens the door to nodes arbitrarily creating infinite amounts of blockchain forks in an attempt to double-spend the currency. A number of solutions to these problems has been suggested, however this falls outside the scope of this report.

A number of cryptocurrency systems such as Peercoin [10] and Nxt [11], among others, utilize a version of a Proof of Stake protocol.

2.4 Smart Contracts

The term Smart Contract was introduced by Nick Szabo [12] and refers to a set of computer protocols and user interfaces intended for formalizing and securing relationships and agreements over computer networks. Smart contracts often capture and translate the clauses of traditional legal contracts into a set of computer logic rules which are executed as any computer program. Smart contract programs are in a lot of cases used to enforce the observation of the contractual clauses by all participating parties [13].

2.5 Cryptographic hash functions

A cryptographic hash function is a function that maps any chunk of data onto a fixed length string of characters. A cryptographic hash function is considered practically irreversible, as in it is computationally infeasible to compute the original message given its hashed value [14].

Such a function must adhere to certain features in order to be considered a good and secure cryptographic hash function.First of all it must provide compression in the sense that the output must be relatively small compared to the input. Moreover it has to be easy to compute a hash of any input value in a small amount of time and the time complexity should not grow rapidly as the length of the input grows. Lastly it must be resistant to collisions, meaning that it has to be computationally infeasible to find any two distinct values that produce the same hash function output. Should such a collision be discovered then the function is considered to be broken.

Similarly to chaotic systems, where a small change in the initial conditions has a significant impact to the output of the system, cryptographic hash functions demonstrate what is called the avalanche effect. According to this effect, the smallest change in the input of the function must result to an extended change of the output. According to the Strict Avalance Criterion, each of the output bits must change with a 50% probability [15].

2.6 Turing Completeness

Turing Completeness is a Computational Theory term used to denote that a system that acts on data can compute any function that the Universal Turing Machine [16] can compute.

According to the Church-Turing thesis [17], any real world computation can be carried out by a Turing machine. That effectively states that no automaton can ever be built that has more computing capability (as to the number of functions it can compute) than the Turing machine [16].

Having that in mind, we can understand that by characterizing a system, such as a programming language, as Turing complete we are basically stating that this system can be arranged in such a fashion that it can perform any real world computation.

At this point we shall note that the concept of Turing completeness does not take into account performance or resource consumption aspects [17]. Naively put, it essentially states that if a there is a solution to a problem and we have a system that can simulate a Turing machine then the solution to that problem will be found by this particular system, in a variable amount of time, having consumed a variable amount of resources.

Chapter 3

The Ethereum Project

Ethereum is an open source project first introduced in 2013, initially described as a "Next-Generation Smart Contract and Decentralized Application Platform". At first glance Ethereum is a peer-to-peer network and an exchangeable cryptocurrency that allows nodes to share computing resources for the execution of programmable smart contracts on the blockchain. There are however multiple different ways to describe Ethereum depending on ones point of view.

In the official guides Ethereum is also described as a 'World Computer', in the sense that it can be seen as a single computing platform which anyone in the world is able to use. In this world computer any number of programs can be encoded and executed, and any participating code can interact and have access to the state of each one of these programs.

In other words, with Ethereum any user can have access to a cheap, zero-infrastructure, global platform that provides a very interesting set of features:

- User authentication, verified by the use of cryptographic signatures.
- Easily deployable payment logic. A payment system can be setup on Ethereum very quickly with no third party reliance.
- Total DDoS resistance. Each application on Ethereum is not executed on any single node; rather it is executed on each and every node on the system. As long as there is one node maintaining the blockchain the application will run perpetually and will be able to be interfaced by any joining node.
- Limitless interoperability. Each Ethereum contract can seamlessly interact with any other contract instance via the provided interfaces in the Ethereum ecosystem
- No server infrastructure. As mentioned before Ethereum is completely built on top of a Peer-to-Peer network with no central server infrastructure involved. Thus, the deployment of an application on the blockchain does not require the setup and the costs of setting and maintaining servers.

Having said this, we can understand that Ethereum strives to provide a platform where anyone can easily deploy and run Internet services.

On a different level, Ethereum can be seen as a facility that enables the creation of smart organizations, i.e. groups of people that want to work together to achieve a goal and have to define and enforce a set of rules. Ethereum aspires to be a the tool used to run anything that falls between a trade agreement between two people and the full orchestration of a government organization. For example, if we imagine that real world objects (e.g. a camera or a lock) can read the Ethereum blockchain, then according to the rules of the running contracts only their rightful owners would be able to use them.

The Ethereum developers claim that they are working towards building the 'Internet as it was supposed to be', as in that their intent is to bring full decentralization back to the Internet. Indeed the Internet was designed as a decentralized system, however it is apparent that it is nowadays concentrated around major hubs such as Internet Service Providers, Cloud Providers, Social Networks, etc. This has led to the phenomenon where the big players of the Internet to enforce their own rules and impose censorship on the distributed content and exclusion of certain parties. In a centralized system users always have to trust the good intentions of some type of authority, be it a government or a company that provides a service. Ethereum strives to eliminate that trust, or rather distribute it among all the participating nodes.

To conclude this introductory section we find it useful to present a concise definition, that captures the essence of Ethereum. This definition was provided by Gavin Wood, one of the pioneers of the project, and describes Ethereum as 'a collection of non-localized singleton programmable data structures'[18].

3.1 Ethereum history

Ethereum was first described by Vitalik Buterin in his article 'Ethereum: A Next-Generation Cryptocurrency and Decentralized Application Platform'[19], in early 2014. Its formal definition was given later that year in Gavin Wood's 'Yellow Paper'[20]. The development of the first implementation started shortly after [21].

On 22 July 2014 the official Ethereum crowdsale was launched [22]. Its purpose was to release the first batch of Ether, the internal cryptocurrency that serves as a transaction cost payment token. The crowdsale ended on 2 September 2014, lasting 42 days [22]. For the first 14 days one could purchase 2.000 Ether for 1 BTC, an amount which was reduced to 1.337 Ether for 1 BTC afterwards. By the end of the sale 60.102.216 Ether was sold to 9.007 buyers for the cost of 31.529 BTC, which by that time had a value of 18.439.000 $ [24, 25].

On July 30 2015 the first official release as well as the first live network, called the Ethereum Frontier, was launched[26]. This can be considered as the Beta testing phase of Ethereum. On this network all purchased Ether can be redeemed. The future releases in the Ethereum network are called, in order, Homestead, Metropolis and Serenity [23].

3.2 Ethereum projects

The statistics regarding the Ethereum blockchain make it apparent that the project has attracted a lot of attention. There is already a number of decentralized applications and concepts built on top of Ethereum, some of which show great potential. We find it useful to present three of these applications that we believe that show great promise as well as demonstrate the power of Ethereum:

- Slock.it: A decentralized physical lock. It involves an Ethereum mini computer as well as an electronic lock that can listen to the blockchain. The concept is that one can lock any asset (e.g. apartment, car, bicycle) behind the 'Slock' and anyone in the world can rent this asset for a fee in Ether. If the transaction is verified then the renting party can unlock the physical asset with their private key and can use it for the duration of the lease. This project very well showcases how Ethereum can connect to the physical world [27].
- BlockApps Strato: A full stack technology solution that allows easy development and deployment of enterprise applications on the Blockchain. It provides a set of tools such as private blockchain ledgers, smart contract development graphical interfaces as well as semi-private P2P networks, i.e. networks that can be isolated and on the same time be able to communicate with the Ethereum network [28].
- Colony: A decentralized platform that enables the creation of decentralized companies across the world. It aspires to bring together people with different skill sets that might want to work together towards a common goal. A reward called 'nectar' is released periodically and people compete for it by contributing ideas, making decisions or performing some sort of work. There is also a community based reputation system that reflects the merit of each individual as well as their impact in the community [29].

3.3 Ethereum implementations

Ethereum has three different officially maintained implementations written in C++ [30], Go[31] and Python[32], all of which are perfectly interoperable over the Ethereum network. More specifically:

- Eth: The C++ implementation; it it said to perform faster than the other implementations and it also is the the basis for the the Mix IDE[33] a contract development toolkit. It comes with a number of network analysis tools like Alethzero[34] and it is suggested as a development platform for Internet of Thins projects. It is also the only client that supports GPU mining [35].
- Geth: The Golang implementation; Geth will be the basis for the future Mist web browser[36]. It is suggested the proper development platform for Web-based applications [35].
- Pyethapp: The Python implementation; Pyethapp is intented mostly for educational purposes. Users wishing to understand the inner workings of Ethereum and are willing to contribute to its expansion are invited to use Pyethapp due to its code readability and clarity. It is not however intended for high-end usage since it lacks in performance [35].

While we could experiment with all three implementations for the needs of this project, we chose Geth for simplicity and to battle time constraints. Thus, any mention of the Ethereum client in the next sections and chapters refers to the Geth implementation.

3.4 Ethereum concepts

Below the main concepts of Ethereum are explained in a theoretical as well as a technical perspective. For the formal definitions of these concepts the reader is invited to read the Ethereum Yellow Paper written by Gavin Wood[20].

3.4.1 Accounts

In Ethereum any entity that holds an internal state is associated with an account, i.e. a private/public key pair. The public key is also considered the address of the account.

Ethereum distinguishes two types of accounts, Externally owned accounts and contracts. An externally owned account is a personal account controlled by a private key. The owner of the private key can send ether or messages to other external accounts. A contract is basically an account that holds its own logic mapped into the code that controls it [37].

3.4.2 Contracts and Transactions

As mentioned in the previous section a contract is an account that contains code, and it is basically Ethereum's way of inserting programmable logic into the blockchain. Contracts, in general, are meant to serve the following purposes [38]:

- Storing the state of values meaningful to other contracts or external entities. For example a cryptocurrency contract can hold the account balances of anyone interacting with that contract.
- Serving as an external account with special access policies. This could be for example a messaging service that only forwards messages if certain conditions are met.
- Mapping and managing relationships among a number of users. For example one could map the logic of a real world financial contract which will always be enforceable within the Ethereum environment.
- Acting like software libraries by providing functions to other contracts. Contracts can interact with each other by passing messages which can contain amounts of Ether, byte arrays or account addresses. When a contract receives such a message it can return data which are consumable by the message sender, thus essentially serving as a traditional function call.

The Ethereum execution environment is inert until something sets it in motion. Transactions are the mechanism through which actions are triggered. Any user can send a transaction from their account to another external account or to a contract. In the first case they can transfer Ether from their balance to the balance of a different user, but other than that these transactions do not have much more interest. In the second case, when the recipient is a contract, the contract wakes up and executes its code.

A contract can read or modify its internal state, consume the received message or in turn trigger the execution of a different contract by sending it a message. When the execution of an action and all of its subsequent actions stops, the environment returns to a halting state until the next transaction is received.

Contracts are written in one of the specialized contract specification languages. These include, Solidity [40] (resembles JavaScript), Serpent (resembles Python) [41] and LLL (resembles LISP) [42]. The code is then compiled into Bytecode which is executed in Ethereum's state machine called the Ethereum Virtual Machine and it is referred to as EVM [43].

3.4.3 Registrars

As mentioned before all entities in Ethereum are associated with an addressable account. Each account is referred to by its 160-bit or 40 hexadecimal character long public key. While this works perfectly for the execution machine, it is rather difficult for the users to remember the addresses of all entities of interest. For that reason a registrar contract, which maps hexadecimal addresses to user specified names, comes hard-coded in the blockchain. By default the client points to the registrar address compiled in the Ethereum network. If a user would like to deploy a different registrar they would have to recompile the contract code and instantiate it again in the Ethereum blockchain.

The above example shows that, while anyone can deploy the same contract multiple times and interact with several of its instances, the value of a contract is defined by its utilization by the network. Anyone can deploy an alternative registrar service and register an arbitrary number of names and addresses on it. However, as long as the network uses the registrar that was deployed first, the alternative service is practically useless.

3.4.4 Ether

Ether is a type of cryptocurrency created to serve as the 'fuel' of the Ethereum network. Its purpose is to be used to pay the network nodes for the amount of computational resources they provide in order to secure the blockchain or execute the contracts. Ether can be obtained either by participating in the mining process, where each mined block is rewarded with 5 Ether, or by purchasing it from a third party.

Ether has a number of denominations the smallest of which is called 'Wei' which equals to 10^{-18} Ether. Bellow you can find a list of the Ether denominations as they relate to Wei:

- Wei: 1
- Ada: 1000
- Fentoether: 1000
- Kwei: 1000
- Mwei: 1000000
- Babbage: 1000000
- Pictoether: 1000000
- Shannon: 1000000000
- Gwei: 1000000000
- Nano: 1000000000
- Szabo: 1000000000000
- Micro: 1000000000000
- Microether: 1000000000000
- Finney: 1000000000000000
- Milli: 1000000000000000
- Milliether: 1000000000000000
- Ether: 1000000000000000000

- Einstein: 1000000000000000000
- Kether: 100000000000000000
- Grand: 10000000000000000000
- Mether: 1000000000000000000000000
- Gether: 100000000000000000000000000
- Tether: 10000000000000000000000000000

It is obvious that the use of all the above names is impractical, thus anything other than Ether and Wei is rarely used.

3.4.5 Gas

The network needs to be rewarded for the amount of computation they provide in order to execute the smart contracts. The amount of Ether paid for each transaction is reflected in the concept of gas.

Each computational activity (e.g. a CPU cycle) costs a certain amount of gas units. The total computational complexity within a transaction defines its final gas expenditure. Each transaction specifies a gas price, i.e. the price the sender is willing to pay for a unit of gas in Wei. The gas price is an incentive for miners to include the transaction in a block. A miner is free to ignore any transaction with a low gas price.

Each transaction specifies a gas limit, i.e. the maximum amount of gas units the sender is willing to expend on this transaction. Since predicting the exact amount of gas each transaction will expend is impossible, the sender must set an upper limit of gas they are willing to pay for the transaction in order to protect themselves from running out of funds (if, for example they invoke an infinite loop). The concept of a gas limit applies to blocks as well and it reflects the sum of the gas spent by all the transactions in a block. Its purpose is to prevent each block from getting too large, which would have an impact on the creation and propagation of the blocks in the network, and by extension the performance of the system.

If the sender of a transaction does not have enough Ether in their account to cover for the transaction then the execution aborts and any intermediate state changes are rolled back to their values before the transaction was send. The gas spend up until that point is subtracted from the account of the sender to cover for the computational costs that occurred before the execution stopped.

To make the concept of gas more clear let us take a look at one example. If a transaction consumes 300 CPU cycles and each CPU cycle costs one unit of gas the transaction simply would cost 300 gas units. If the gas price is set to $5x10^{10}$ Wei (the default value in Ethereum client) then the whole transaction would cost $1.5x10^{12}$ Wei ($1.5x10^{-6}$ Ether). If the sender of the transaction had specified the gas limit to be for example $1.4x10^{12}$ Wei, the transaction would have been aborted as soon as it would have expended that value. Yet the sender would have to pay that amount.

Table 3.1 shows the gas fees for each type of operation as they are presented in the ether.fund website [44]:

Operation	Gas	Description
step	1	Default amount of gas to pay for an execution cycle
stop	0	Nothing paid for the STOP operation
suicide	0	Nothing paid for the SUICIDE operation
sha3	20	Paid for a SHA3 operation
sload	30	Paid for a SLOAD operation
sstore	100	Paid for a normal SSTORE operation (doubled or waived sometimes)
balance	20	Paid for a BALANCE operation
create	100	Paid for a CREATE operation
call	20	Paid for a CALL operation
memory	1	Paid for every additional word when expanding memory
txdata	5	Paid for every byte of data or code for a transaction
transaction	500	Paid for every transaction

Table 3.1: Amount of gas units to be paid for each EVM operation. Source: `http://ether.fund/tool/gas-fees`

3.4.6 Mining

Ethereum much like most blockchain technologies uses a mining process to secure the network. The process involves the production of blocks whose validity will be verified by the network. Just as the Bitcoin protocol, a block is only valid if it contains a Proof of Work of a certain difficulty.

The PoW algorithm used by Ethereum is called Ethash [45] and it involves finding a nonce input to the algorithm so that the result is below a certain threshold depending on the difficulty. The mining difficulty is automatically adjusted so that a block is produced every 12 seconds.

Ethash is designed to be memory hard in order to be impossible to implement on ASIC(Application-Specific Integrated Circuit) machines. This is because in order for the PoW to be calculated, a subset of a fixed resource depending on the block header and the nonce. This resource, a Directed Acyclic Graph (DAG), is 1GB in size and it is different every 30000 blocks. With 12s block time 30000 blocks amount to a 100 hours period, called an 'epoch'. This DAG depends solely on the height of the blockchain and it can be, thus, pre-generated. Its generation takes a few minutes and no block can be produced before the DAG is generated. However, the full graph is not needed for block verification since the appropriate subsets can be calculated from a 16MB cache, requiring low CPU power and memory [39].

The miners receive a reward for the computational effort they invest on the network. The amount they receive includes a static reward of 5 Ether for the discovered block and all the gas expended within the block, i.e. the gas consumed by all the transactions in that block. The static reward is issues as a transaction from the miners themselves while the transaction costs are paid by each transaction sender. It is expected that as the network grows, the transaction gas within each block will surpass the value of the static rewards and will be the main incentive for mining [39].

Interestingly enough, Ethereum also rewards blocks in a forked chain called 'Uncles'. An uncle is a block that is an ancestor of the current block but it is not part of the main (longest) chain. For example if we are currently on block number 30 its parent is block 29 whose parent is block 28. If there is another block that points to block 28, that block is an uncle of block 30. A block is considered an uncle only if it is up to 6 blocks back from the current block. The reward for a valid uncle is 7/8 of the static block rewards, that is 4.375 Ether. A maximum of 2 uncles per block is allowed [39].

Ethereum chooses to reward uncles in order to neutralize the effect of mining rewards being collected by one central institution, due to network lag. Moreover, this increases security by including more work in the Blockchain. For a full grasp of the incentive behind rewarding uncles the reader is advised to read Vitalik Buterin's article 'Toward a 12-second Block'[46] and Yonatan Sompolinsky's and Aviv Zohar's paper 'Accelerating Bitcoin's Transaction Processing'[47].

At the moment, the mining difficulty on the live Ethereum network is such that only GPU hardware can perform this action. CPU mining is possible but the probability that any real Ether can be mined that way is minimal [48].

3.5 Solidity

Solidity is one of the high level programming language used to describe smart contracts. Solidity is the language that is officially maintained by the Ethereum project and suggested in the guides as the main contract language. Solidity is Object Oriented and it resembles JavaScript. Below we will present some of its main features.

3.5.1 Types

Solidity supports a number of different data types. Like any traditional language it supports booleans, integers and strings. There is no support for floating point variables as of yet [50].

A very interesting data type is that of an Ethereum address. It holds the 20 byte representation of an Ethereum account address, be it an external account or a contract. The address data type has internal predefined members to check the balance of an account or transfer Ether via a contract, as well as to call functions from other contracts [50].

Solidity also supports structs and enumerations as well as byte arrays that can hold data of any type. Moreover, solidity support mappings which are in essence key-value stores that map keys of any data type to values of any data type as well. An accessor function is created for each declared variable.

3.5.2 Events

Events are the way Solidity provides in order for accessing the transaction logs, a special data structure in the Blockchain. Functions can emit these events populated with return values, and event messages will be broadcast and stored on the blockchain. Each application has to specify certain JavaScript callback functions that listen for these event messages and print them to the user interface. Event messages are not accessible from within contracts, not even the contracts that have created them [51].

3.5.3 Functions

Solidity distinguishes two types of functions, constant and transactional. Constant functions are the those functions whose purpose is to return a value and cannot update the state of the contract. They can be called directly and do not consume gas since they do not modify the blockchain. Transactional functions are used to modify the state of the contract. Whenever called an amount of gas has to be supplied to cover the transactional costs. Constant functions need to be declared as such by using the keyword 'constant' [51].

There are four levels of visibility for Solidity functions. These are:

- External: External functions are part of the contract specification and they can be called by other contracts, but they are not accessible by the contract it self.
- Public: Public functions can be called by the contract itself or by any external contract or entity.
- Internal: Internal functions can only be accessed by the contract itself and its derivative (inherited) contracts.
- Private: Private functions are visible only to the contract itself and cannot be called by any external entity or derivative contract.

3.5.4 Function Modifiers

Function Modifiers are constructs used to change the behaviour of a function. They are mainly used to check if a condition is satisfied before a function can be executed. Modifiers are inheritable properties of functions and each function can belong to multiple modifiers [51].

3.6 Building a test network

Each Ethereum release is accompanied by a network of peer nodes. On this network computation has to be paid for with Ether that has to be either purchased or mined on the live blockchain. However, if one wishes to try the technology and build a decentralized Ethereum application in a test environment without spending 'real' Ether or expending large computational resources in order to mine it, they can build a private network, isolated from the live Ethereum network. On a private network the initial block difficulty can be adjusted in such a way that CPU mining can produce blocks in small time. Any Ether produced cannot be redeemed in the live network since it was mined on a different blockchain. However it can be used to cover transaction costs in the same network.

For the needs of this project such a network was initiated, comprising of 3 peer nodes, an OS X 10.11 laptop, an Ubuntu 15.04 workstation and a Debian 8.3 server. All the nodes have installed the Geth Ethereum client. The peers can join the network by invoking the following command:

```
# geth \\
--networkid <id> --datadir . \\
--genesis /path/to/genesis.json \\
--nodiscover console \\
2>>geth.log
```

The last part of the of the command is to redirect the large amount of log entries produced, from the interactive shell towards an external file.

By default the peer discovery mechanism is on, and the peers find each other via whispering. However, since the Ethereum client comes with a set of hard coded static peers, which will try to connect to the private network, we turn off the discovery with the 'nodiscover' flag. In that case we will have to add each peer from the interactive shell by the following command:

```
>admin.addPeer("enode://pubkey@ip:port")
```

The 'enode' parameter is composed by the public key of the node, the IP address of the host and the TCP port where the Ethereum client is listening at. The Ethereum address of each node can be obtained by running the following (with an example output):

```
>admin.nodeInfo.enode
"enode://99e11e03aa79ef746957add61d3017053ebea
142e3c147d32aa4fa9ddf7df7ff2ec85fdb3e09cd5c9cd
cd6ba88d6ff81e20ec0740c612a4ddc016dff598acbe5
@192.168.1.1:30303"
```

In order for the nodes to start syncing they will have to belong to the same network ID and have a matching genesis block. While the genesis block for the live network has to be produced by a python script [49] in a private network the genesis block can be any valid block with no parent. For example:

```
{
        "nonce": "0xcafebabeb1aab1aa",
        "timestamp": "0x0",
        "parentHash": "0x0000000000000000000000000000
        0000000000000000000000000000000000000000",
        "extraData": "0x0",
        "gasLimit": "0x8000000",
        "difficulty": "0x400000",
        "mixhash": "0x0000000000000000000000000000
        0000000000000000000000000000000000000000",
        "coinbase": "0x3333333333333333333333333
        333333333333333",
        "alloc": {
        }
}
```

While the above make the procedure seem relatively simple, the amount and quality of the documentation and official guides provide a rather steep learning curve and one could easily run into frustrating issues. For example, the guides suggest that the ID of a private network should be large enough to avoid collisions. However it is not mentioned that there is a maximum value that the network ID can have. A greater value will cause the peers not to communicate. A safe ID should fall within the range of 99-99999.

Moreover, one should be careful not to include a too small initial difficulty in the genesis block. In such a case blocks would be mined rapidly by all the nodes in the network without allowing them to sync between each discovered block. That can lead to the phenomenon that a node will constantly be mining on a stale fork of the blockchain and will be eventually be kicked out of the network as a 'useless peer'.

Chapter 4

TheDistributed Application

As has been mentioned before the main purpose of Ethereum is the deployment of fully decentralized smart contracts. Thus, we developed such a smart contract in Solidity, Ethereum's contract representation language. The decentralized application we decided to build simulates the procedure of a civil court of law.

In certain jurisdictions civil law cases are tried according to the adversarial system which states that the case is discussed in front of an impartial person (judge) or group of people (jury) and the party that claims their case in the best way, according to the opinion of the impartial judging body, wins the case. This is opposed to the inquisitorial system that is applied during criminal law cases, according to which the court of law attempts to establish absolute facts that will determine the outcome of the case.

Our application is based on such a system; in its essence one party decides to call an opponent to the court and debate a case. The side with that persuades the jury about the superiority of their claims, wins their votes and by extension the case. The allowed level of participation for each entity in the system is reflected in an exchangeable token that we will call 'justice'. Moreover, we suggest the collection of an amount of Ether from the judges and jurors to serve as a collateral, ensuring that they will not leave the case without participating.

At this point we have to absolutely point out that the author of this report does not in any case advocate the transition of the administration of justice from the, properly educated in jurisprudence, legal entities to anonymous Internet users, or any other form of faceless body of people for that matter. This type of application was selected solely due to the fact that it can very well showcase the capabilities of Ethereum. It involves the creation of a new cryptocurrency, the implementation of the operational rules of a democratic organization as well as the digitization of a real world administrative procedure. All the above are domains in which Ethereum's contribution can have great impact.

4.1 The Contract in Natural Language

The logic of developed smart contract can be captured in the following natural language clauses:

1. The plaintiff creates and initiates the case.
   - 1.1 Appoint a treasurer.
   - 1.2 Appoint defendant.
   - 1.3 Provide the address of the exchangeable 'justice' token.
   - 1.4 Set initial variables (description, debate rounds, etc.).
2. Opposing parties set the bench by appointing judges.
   - 2.1 If bench is full no more judges can be added.
   - 2.2 If there is an attempt to add a judge by an entity that is not plaintiff/defendant judge will not be added.
   - 2.3 No judge can be added twice.
3. Jurors add themselves to the jury.
   - 3.1 If jury is full no more jurors can be added.
   - 3.2 A juror cannot be added twice.
4. All parties must request their preallocated amount of 'justice'.
   - 4.1 Jurors and judges must send a predefined amount of ether to the treasurer as a participation collateral.

5. Opposing parties state their arguments exchanging turns over a number of debate rounds.
    5.1 The debate process cannot start if all the participating parties have not assumed their roles.
    5.2 No party can speak out of turn.
    5.3 Each argument costs one unit of 'justice'.
    5.4 At the end of each round the turn is toggled to the other party.
6. Judges can interfere at any point and provide their stance on the submitted party arguments.
    6.1 Each judge interference costs one unit of 'justice'.
7. Jurors vote on the outcome of the case.
    7.1 Jurors cannot vote if the debating process has not been concluded.
    7.2 Each vote costs one unit of 'justice'.
8. Votes are counted and winning party is announced
    8.1 Counting procedure cannot start if the voting has not concluded
9. Collateral and rewards are released to the bench and jury.
10. Case is considered closed.

4.2 The Justice Token

For the purposes of controlling the flow of the procedure and ensuring that each participating entity does not exceed their allowed participation limit (e.g. a juror can only cast a single vote), we have created a cryptocurrency to be distributed as a token among the court roles. Each taken action within the contract costs one unit of the justice token.

4.3 Contract entities

The different entities that participate in this contract are described below:

- Plaintiff: The plaintiff is the party that has a claim over the opposing party. In our case the plaintiff initiates the case and sets the configurable parameters of the contract.
- Defendant: The defendant is the party that opposes the plaintiff. They have the right to appoint one or more judges. The amount of justice given to both parties equals the number of the debating rounds.
- Bench: The bench is the body of the judges. In this particular implementation of the case contract the role of a judge might seem unnecessary since they only have the right to interfere at any point without their interference actually affecting the procedure.
- Jury: The jury votes upon the outcome of the case once the debating has concluded. In our implementation each member of the jury has only a single vote (thus, one token of justice) and there is no configurable level of agreement between the jurors. Simple majority will win the case. Moreover, there is no check to make sure that the number of jurors is always an odd number, thus a tie is a possible outcome.
- Treasurer: The role of the treasurer might also seem unclear in the current implementation. The general responsibility of the treasurer is to collect the collateral and distribute the 'justice' token.

4.4 Public Functions

The following methods are the publicly invokable functions that allow users to interact with the contract either to read or modify its internal state.

- Case: The contract is simply called 'Case' and the function that has the same name as the contract is its constructor

method. The method is invoked in order for the contract to be instantiated in the blockchain, and it is called with a set of initial parameters. This parameters are bench size etc. When invoked correctly it emits a 'caseInitiated' event.

- newJudge: This method can only be invoked by the opposing parties and it used to appoint new judges. Either the plaintiff or the defendant must call the function supplying the public address of a personal account. Upon successful execution the supplied account is added to the bench of judges and a 'judgeAdded' event is emitted. In case that one of the addition rules is violated (e.g. bench is full, or judge already added) a 'judgeNotAdded' event is emitted.
- newJuror: Any account that wishes to participate as a juror can add themselves to the jury body, provided that they do not already serve as a judge or juror, they are not one of the opposing parties and the maximum number of jurors has not been reached. Upon successful invocation a jurorAdded event is emitted. A jurorNotAdded event is emitted otherwise.
- requestJustice: All the participating parties, except the treasurer must request for their preallocated amount of 'justice' token to be sent to them. In the case that the requestor is either a judge or a juror, they must send a predefined amount of ether to the contract as a collateral. A justiceSent or justiceNotSent event is emitted, according to the result of the method.
- partySpeak: This method is called by either the plaintiff or the defendant when they need to present their arguments. There are internal checks to ensure that only the opposing parties can submit arguments for the case. There is also an internal check to ensure that parties cannot submit arguments out of turn. The method raises a partySpoke or an outOfTurn event in the cases of success and failure respectively. The method can only be called after all the participating parties have assumed their role.
- judgeSpeak: This method can only be called after the debatign procedure is ready to start and only by the judges, whenever

they wish to interfere to the case. It emits a judgeSpoke or a judgeMute event depending on the outcome of the call.

- juryVote: Each juror votes on the outcome of the case by calling this function and submitting 0 in favor of the dedendant and 1 in favor of the plaintiff. It emits a juryVoted event in the case of success and a juryNotVoted event otherwise. This method can only be invoked after the debating process has concluded.
- caseClosed: The final method is called only after the voting has lapsed and its purpose is to count the votes and annouce the winner of the case. It will emit a plaintiffWon or defendantWon or tied event depending on the outcome of the case.

4.5 Limitations and Proposed Extensions

Our implementation of the contract provides certain limitations that prevents the contract to be executed as designed. We will present some of them briefly below:

- The collateral collection mechanism does not work due to an implementation bug. This halts the contract execution due to a modifier function checking if the collateral has been collected.
- There is no check if the amount of collateral sent is the predefined amount.
- There is no control over the validity of a vote. If a juror sends a number 1 value the vote is added to the plaintiff and any other value is added to the defendant.
- There is no control in place to check if one party is already participating as a different role (e.g.The plaintiff can be a juror as well)
- Anyone can tamper with the justice balance of another entity. The justice balance is not a locked wallet.
- A party can fill up the bench by themselves without giving the other party a chance to appoint a judge.

However most of these limitations can be overcome with little effort by additional checks injected sporadically in the code

Moreover we can suggest certain extensions to the design of the contract that will make it more meaningful:

- A more active role can be given to the judges. For example a judge can decide if the case has a basis to go forward or it will be shutdown.
- A rewarding system can be put in place in order to distribute rewards to the parties according to the participation, while on the same time it ensures a fair trial.
- We can ensure that the outcome cannot be tied by enforcing an odd number of jurors. Moreover we can configure the level of consensus to be reached within the jury for the outcome to be accepted (e.g. Simple majority, 2/3 majority, unanimous verdict, etc.)
- More meaning should be given to the justice token. Now it is just a number being passed between the parties and the treasurer. In a future implementation the token could have a more active role in the process.

4.6 Setup scripts

In order to make the application more complete we wrote a number of scripts in JavaScript that setup the execution environment, by creating and registering accounts for the participating entities, distributing Ether among them as well as automatically deploying and mining the contracts. Moreover, several callback functions have been defined in order to monitor the platform for incoming events and respond to them.

Chapter 5

Evaluation

In this chapter we will present the experiences we gathered, using Ethereum from an operational as well as a development point of view.

Generally, Ethereum has brought some extremely powerful concepts into life and can certainly be a huge leap forward towards the decentralization of the Internet. However, it is still under heavy development and suffers from low maturity and lack of proper documentation. It demonstrates certain performance and stability issues and the security status of the platform is largely uncertain. Indeed, the official guides warn strongly against running anything production ready on the current version of Ethereum.

Below we share some of our insights regarding different aspects of our evaluation.

5.1 Difficulty

There is a clear imbalance between the difficulty of setting up and operating a development environment and actually developing an Ethereum application. Ethereum consists of several different components and its foundation lies in complex concepts rooted in cryptography, mathematics, economics, etc. While a deep understanding of these concepts is not mandatory for setting up an Ethereum client or a private network of peers, the user a lot of times has to rely on outdated or unclear guides, spread around different sources that sometimes conflict with each other. Moreover, the guides often reference other sources that have been abandoned or deleted.

Developing, on the other hand, is much more intuitive since it largely resembles traditional object oriented programming. Although the solidity tutorial is itself also not completed, and sometimes introducing fundamental concepts for the first time in the FAQ section, it is relatively easy for anyone with some programming experience to rapidly develop and deploy complex smart contracts. However, the debugging and release process is not as intuitive as one might be used to. Each change in the code involves redeploying a contract instance and waiting for it to be mined in the Blockchain before the developer can interact with it again. This can sometimes be counterproductive.

A measure of the aforementioned imbalance between development and operations is the fact that it took nearly 2 weeks to get to a point where we have a network of nodes syncing with each other correctly and we are at a position to understand why everything works the way it does. In contrast it took roughly 3 days to have an initial yet fully functional version of the contract we described in chapter 4.

5.2 Computational Power and Storage

We have mentioned before that the Ethereum network discovers new blocks every 12 seconds. This means that there is a release of a burst of computational power each time the a block is discovered and the state of the Blockchain is updated. While this is an acceptable waiting time for a number of applications, such as every day financial transactions (transferring funds can take a few days if a foreign bank is involved for example), it is forbidding for certain time critical applications such as industrial operations or health procedures.

Moreover, a big scalability issue becomes apparent if we think that the state of the blockchain has to be stored in each individual client. If Ethereum potentially grows to support thousands or millions of applications this model cannot work.

There is effort invested into solving the issues above by adding new elements in the Ethereum ecosystem. These include Swarm, a bittorrent-like P2P storage system [52], Whisper, a low level messaging application [53] and Mist a front-end web browser [36]. These components are at a Proof of Concept stage and have not been used at all during this project.

5.3 Cost

In the third chapter we described the concept of gas by giving an example. Let us revisit this example specific to our contract. Our contract compilation costed 2.8×10^6 units of gas to compile. Multiplying it with the standard gas price we have $2.8 \times 10^6 \times 5 \times 10^{10} = 1.4 \times 10^{17}$ Wei = 0.14 Ether. According to the price of Bitcoin at the time of the Ether sale this amount is roughly 0.45$. Thus compiling a medium sized contract costs a around 0.5$ depending on the gas price. However, each transaction that interacts with the contract costs a fraction of this price.

5.4 Security

An interesting security question might arise. Can malware be built in the Blockchain? The intuitive answer is that this is not possible. While Ethereum can map anything that can be programmed into a Blockchain smart contract, smart contracts are applications that hold enforce certain logic and relationships between entities but in contrast to other programming models, smart contracts do not handle files or initiate network connections.

However, the security of the whole platform depends on the security of the Ethereum Virtual Machine (EVM) and each client implementation. In theory if there is an implementation flaw in the Ethereum client a contract could try and print an executable command that will actually have impact on the client (e.g. initiate an Ether transfer from an unlocked account). Moreover if the EVM implementation has exploitable issues one could attempt to break out of the contract execution environment and attempt to inject code in the memory of the host machine with destructive effects.

Furthermore, it is possible to see the executable bytecode representation of the contract but the high level source code is not always available so a contract might claim to perform an action while in reality it does something else. Even if the contract code and the version of the compiler through which its executable was produced, is known there is always a level of trust involved.

Even so, a contract can very openly do something illegal, such as spreading threats or classified content or even byte representations of child pornography imagery. Due to the lack of proper identification (the contract initiator is merely a 40 character address) and a central controlling authority, all the nodes in the network can potentially host illegal content.

Chapter 6

Conclusion

In this chapter we will present our conclusions in regard to how they answer our research questions.

Our main research question was: Can Ethereum be directly used to rapidly deploy meaningful and sufficiently performing trusted applications with added value over traditional approaches?

We can argue that the answer, in general, is yes. We have demonstrated that a full decentralized application can be developed and launched with minimal effort to the Ethereum network. The application we have developed is much more than a trivial 'Hello World' project and it is fully decentralized which adds great value.

More specifically:

Can Ethereum be used to deploy non-trivial applications?

As we said before the application we built involves features, such as transferring of funds or public verifiability of votes, which in traditional approaches are too complex and very difficult to solve. Thus the answer is a clear yes.

Can we achieve acceptable application performance even with the currently present constraints?

If we can look past the 12 second block time problem, we can achieve performance which for a large number of applications is totally acceptable.

What is the typical time needed for an Ethereum application to be developed and launched?

We are not in a position to give a totally clear answer to this question, since we can only estimate the mean deployment time. An application of course can be launched instantly after it is developed. Development time, however, depends heavily on the experience of the developer and complexity of contract in question. However, since there is no infrastructure involved, the overhead of having to setup servers and database systems is waived, which greatly speeds up the process.

What is the added value of using Ethereum over a more traditional development approach?

The added value of Ethereum becomes apparent by the fact that every application is decentralized and is controlled by no single authority. In our court of law example, in a traditional client-server approach one would have to trust the operational authorities for ensuring the security of the system and not tampering themselves with the data. In the case of Ethereum this trust is eliminated or rather it is distributed among the nodes.

Chapter 7

Future Work

This project stands between fully investigating the technical inner workings of Ethereum and looking at the platform from an abstract, application development point of view. There is a great amount of research to be done all aspects of Ethereum and a large number of projects can be spawned in order to extend the present work.

First of all, for the needs of this project we only used a subset of all the features offered by Ethereum. One could look much deeper into the implementation of the project and use all features and tools (such us the interfacing capabilities between applications) available to leverage even more the power of Ethereum. Moreover, since only the Golang implementation was used we could also get involved with the other client implementations and evaluate the additional features they offer.

Apparently we can build upon the already developed contract to overcome its limitations and further extend it to provide bigger functionality. Moreover, we could complement it with a proper web-based user interface, which would allow us to evaluate JavaScript API provided by Ethereum.

In a more technical sense we could also attempt a proper performance benchmarking and cost evaluation for a set of different applications. Furthermore, since the security of the platform is largely uncertain, a lot of effort could be invested into discovering and potentially fixing its vulnerable components.

Lastly, Ethereum can be the basis of great research in the social sciences fields. By developing a number of different contracts, regarding crowdsourcing, alternative economies, and democratic organizations, one could design a number of very interesting and useful social experiments.

Bibliography

[1] Rooney, B. (2016). Bitcoin worth almost as much as gold. [online] CNNMoney. Available at: http://money.cnn.com/2013/11/29/investing/bitcoin-gold/index.html

[2] Antonopoulos, A. (2015). Mastering Bitcoin. Sebastopol, CA: O'Reilly.

[3] Ethereum.org, (2016). Ethereum Frontier. [online] Available at: http://ethereum.org

[4] Rooney, B. (2016). Bitcoin worth almost as much as gold. [online] CNNMoney. Available at: http://money.cnn.com/2013/11/29/investing/bitcoin-gold/index.html

[5] Bitcoin.org, (2016). FAQ - Bitcoin. [online] Available at: https://bitcoin.org/en/faq

[6] Pristley T. "Bitcoin Declared An Inescapable Failure". Forbes.com, (2016)[online] Available at: http://www.forbes.com/sites/theopriestley/2016/01/15/bitcoin-declared-an-inescapable-failure/#1bdc02749bf8

[7] Nakamoto, Satoshi (October 2008). "Bitcoin: A Peer-to-Peer Electronic Cash System" Available at: https://bitcoin.org/bitcoin.pdf

[8] Dwork, Cynthia; Naor, Moni (1993). "Pricing via Processing, Or, Combatting Junk Mail, Advances in Cryptology". CRYPTO'92: Lecture Notes in Computer Science No. 740 (Springer): 139–147.

[9] Jakobsson, Markus; Juels, Ari (1999). "Proofs of Work and Bread Pudding Protocols". Communications and Multimedia Security (Kluwer Academic Publishers): 258–272.

[10] Peercoin.net, (2016). Peercoin. [online] Available at: https://www.peercoin.net

[11] Nxt.org, (2016). Nxt.org | Decentralized Financial Ecosystem. [online] Available at: http://nxt.org

[12] Szabo, N. (1997). Formalizing and Securing Relationships on Public Networks. First Monday, 2(9).

[13] Szabo.best.vwh.net, (2016). Secure Property Titles with Owner Authority. [online] Available at: http://szabo.best.vwh.net/securetitle.html

[14] Stamp, M. and Low, R. (2007). Applied cryptanalysis. Hoboken, N.J.: Wiley-Interscience.

[15] Webster, A. F.; Tavares, Stafford E. (1985). "On the design of S-boxes". Advances in Cryptology - Crypto '85. Lecture Notes in Computer Science 218. New York, NY,: Springer-Verlag New York, Inc. pp. 523–534

[16] Sipser, M. (1997). Introduction to the theory of computation. Boston: PWS Pub. Co.

[17] Lewis, H. and Papadimitriou, C. (1981). Elements of the theory of computation. Englewood Cliffs, N.J.: Prentice-Hall.

[18] Ethereum.gitbooks.io, (2016). What is ethereum? | Ethereum Frontier Guide. [online] Available at: https://ethereum.gitbooks.io/frontier-guide/content/ethereum.html

[19] Buterin, V. (2014). Ethereum: A Next-Generation Cryptocurrency and Decentralized Application Platform. [online] Bitcoin Magazine. Available at: https://goo.gl/yrrUFm

[20] Wood, G., ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER. (2014). [online] Available at: http://gavwood.com/paper.pdf

[21] Ethereum, (2014). C++ Code+Build FAQ. [online] Available at: http://forum.ethereum.org/discussion/170/c-code-build-faq

[22] Ethereum Blog, (2014). Launching the Ether Sale - Ethereum Blog. [online] Available at: https://blog.ethereum.org/2014/07/22/launching-the-ether-sale

[23] Ethereum Blog, (2015). The Ethereum Launch Process - Ethereum Blog. [online] Available at: https://blog.ethereum.org/2015/03/03/ethereum-launch-process/

[24] Ether.Fund, (2016). Ether Price & Market for Ethereum | Ether.Fund. [online] Available at: http://ether.fund/market

[25] Ethereum Blog, (2014). Ether Sale: A Statistical Overview - Ethereum Blog. [online] Available at: https://blog.ethereum.org/2014/08/08/ether-sale-a-statistical-overview/

[26] Ethereum Blog, (2015). Ethereum Launches - Ethereum Blog. [online] Available at: https://blog.ethereum.org/2015/07/30/ethereum-launches

[27] Slock.it, (2016). Slock.it. [online] Available at: http://slock.it

[28] BlockApps | Strato, (2016). Blockapps Blockchain Development Tools. [online] Available at: http://www.blockapps.net

[29] Colony.io, (2016). Colony. [online] Available at: http://colony.io

[30] Isocpp.org, (2016). Standard C++. [online] Available at: https://isocpp.org

[31] Golang.org, (2016). The Go Programming Language. [online] Available at: https://golang.org

[32] Python.org, (2015). Welcome to Python.org. [online] Available at: https://www.python.org

[33] GitHub, (2015). ethereum/wiki. [online] Available at: https://github.com/ethereum/wiki/wiki/Mix:-The-DApp-IDE

[34] GitHub, (2016). ethereum/alethzero. [online] Available at: https://github.com/ethereum/alethzero.

[35] Ethereum.org, (2016). Install the Command Line Tools. [online] Available at: https://ethereum.org/cli

[36] GitHub, (2016). ethereum/mist. [online] Available at: https://github.com/ethereum/mist

[37] Ethereum.gitbooks.io, (2016). Account types and transactions | Ethereum Frontier Guide. [online] Available at: https://ethereum.gitbooks.io/frontier-guide/content/account_types.html

[38] Ethereum.gitbooks.io, (2016). Contracts and transactions | Ethereum Frontier Guide. [online] Available at: https://ethereum.gitbooks.io/frontier-guide/content/contracts_and_transactions_intro.html

[39] Ethereum.gitbooks.io, (2016). Introduction | Ethereum Frontier Guide. [online] Available at: https://ethereum.gitbooks.io/frontier-guide/content/mining.html

[40] Ethereum.github.io, (2016). Solidity, Ethereum Smart Contract Programming Language. [online] Available at:http://ethereum.github.io/solidity/

[41] GitHub, (2015). ethereum/wiki. [online] Available at: https://github.com/ethereum/wiki/wiki/Serpent

[42] Ethereum, (2014). LLL. [online] Available at: https://forum.ethereum.org/categories/lll

[43] GitHub, (2015). ethereum/wiki. [online] Available at: https://github.com/ethereum/wiki/wiki/White-Paper

[44] Ether.Fund, (2016). Gas Fees for Ethereum Operations. [online] Available at: http://ether.fund/tool/gas-fees

[45] GitHub, (2015). ethereum/wiki. [online] Available at: https://github.com/ethereum/wiki/wiki/Ethash

[46] Ethereum Blog, (2014). Toward a 12-second Block Time - Ethereum Blog. [online] Available at:https://blog.ethereum.org/2014/07/11/toward-a-12-second-block-time

[47] Sompolinsky, Y. and Zohar, A. (2013). Accelerating Bitcoin's Transaction Processing Fast Money Grows on Trees, Not Chains. Available at: https://eprint.iacr.org/2013/881.pdf

[48] Ethereum.org, (2016). Get Ether. [online] Available at: https://ethereum.org/ether

[49] Ethereum Blog, (2015). Final Steps - Ethereum Blog. [online] Available at: https://blog.ethereum.org/2015/07/27/final-steps

[50] Solidity.readthedocs.org, (2016). Types – Solidity 0.2.0 documentation. [online] Available at:http://solidity.readthedocs.org/en/latest/types.html

[51] Solidity.readthedocs.org, (2016). Contracts – Solidity 0.2.0 documentation. [online] Available at: http://solidity.readthedocs.org/en/latest/contracts.html

[52] Ethereum, (2015). Swarm. [online] Available at: https://forum.ethereum.org/categories/swarm

[53] GitHub, (2014). ethereum/wiki. [online] Available at: https://github.com/ethereum/wiki/wiki/Whisper

Appendix: Source code

All the Source Code developed for the needs of this project, including the Solidity Smart Contracts as well as the JavaScript setup scripts can be found in the following github repository.
`https://github.com/ntrianta/rp2ethereum`