Interplanetary Smart City

Marcel Zak

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Master of Engineering
of the
University of Aberdeen.

1 4 9 5

Department of Computing Science

2018

Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed:

Date: 2018

Abstract

Will be here

Acknowledgements

Thanks Mum!

Contents

Chapter 1

Introduction

The challenges of cities are changing. As we use more of that finite resource of clean drinking water, as we create more waste and use more energy, we must think very differently how to solve the problems. In 2014 United Nations estimated that 54 percent of the world's population lived in urban areas and predicted that the number increases to 66 percent by 2050 [1]. Air pollution in cities is proliferating. World Health Organization estimated that in 2012 died 3.7 million people because of outdoor air pollution exposure [2]. What is a smart city? Is it a solution to these problems? In 2018 there is no agreed definition of a smart city. I would say that it is an urban area that produces and uses data from many sensors called Internet-of-Things (IoT) devices. These data collections help to improve life, health, comfort and resource management of the city. The answer is that it is not the solution to all problems that cities have. On the other hand, it can help to use resources more efficiently and bring new insights based on collected data to issues we are facing. One example can be an improvement in public transportation and traffic light control. This improvement can directly decrease air pollution. The same approach can be used in our homes, villages or even on a global and interplanetary scale. In 2024 first crew should begin their mission to Mars[1] and set up first Mars base, from which we can build a city and eventually a self-sustaining civilisation. An interplanetary smart city will require communication technology that is secure, scalable, distributed and aware of the physical distance between information and request location.

This dissertation reports the development, testing and evaluation of Interplanetary Smart City (IPSC), a fully decentralised peer-to-peer (P2P) application designed to allow communication between IoT devices. This application can be deployed in a variety of scenarios from smart homes to interplanetary smart cities. The project aims to explore the possibilities of utilising blockchain technology, the best ideas from multiple (P2P) protocols and ideology of IPFS [3]. Such an application will bring numerous advantages in comparison to traditional cloud-based solutions. IPSC ref will allow secure, robust, reliable and space aware communication without a central point of failure and possible savings on server hosting and cloud services.

1.1 Motivation

Data are, supposedly, the currency of the Internet age. Companies are increasingly allowing payments for their digital services with information rather than money [4]. Existing technologies

[1]http://www.spacex.com/mars

that are used in smart cities are mostly centralised and do not allow easy trade with collected data between other parties. This approach worsens all mentioned criteria for security, reliability, robustness and no physical distance awareness. On the other hand, it is beneficial to the companies providing these services because they have easy and usually free access to the data collections. In order to use such services as Oracle Cloud IoT[2], Google Cloud IoT[3], Salesforce IoT[4] or Microsoft Azure IoT[5] our device must be connected to the Internet. This dependency can be a disadvantage because in a case of Internet connection failure we cannot access our data. Furthermore, if the service provider is hacked or experiencing technical issues, then our data collections can be stolen, inaccessible or permanently deleted. This poses high dependency on the service provider and Internet connectivity. One example of service provider dependency can be Logitech that decided to intentionally brick all Harmony Link devices remotely on 16 March 2018 [5].

The next issue is that IoT devices need to communicate each other often. The standard client-server model can introduce a bottleneck and increased latency. Moreover, it is a single point of failure. On the contrary, P2P network is ideal for a smart city. The workload is spread across multiple devices and requested data can be retrieved directly from the closest local node that is in possession of them. In the case of a future interplanetary city, the physical distance of required data is critical. The time required to travel radio wave a distance between Earth and Mars is approximately from 4.3 minutes up to 21 minutes. This depends on the position of the planets. Furthermore, interplanetary space is a very different environment. High-energy ionising particles (electrons, heavy ions and protons) of the space environ causes Single Event Effects (SSE) such as Single Event Upset, Single Event Transient, Multiple Bit Upset and many other destructive and nondestructive SSE. They are responsible for the arbitrary behaviour of electronics and corruption of memory [6]. From these examples, it is clear that the current approach of IoT and a smart city communication is not suitable for the future use.

1.2 Objectives

The main research interest in this project is focused on the applications of P2P, Byzantine fault tolerant Blockchain Technology for an interplanetary smart city. The main project objectives are to develop, test and perform an initial evaluation of an application prototype that allows scalable, secure, robust, reliable and information distance aware communication for IoT devices. The project addresses a subtask of providing an easy way of data collection exchange between untrusted parties. (The goal of the IPSC research is, therefore, to enable...)

[2] https://cloud.oracle.com/iot

[3] https://cloud.google.com/solutions/iot/

[4] https://www.salesforce.com/products/salesforce-iot/overview/

[5] https://www.microsoft.com/en-us/internet-of-things/azure-iot-suite

Chapter 2

Background & Related Work

This chapter is dedicated to background and work related to this project. The chapter covers advantages and disadvantages of several peer-to-peer mechanisms and current approach for communication between IoT devices in a smart city. Moreover, high-level description of blockchain mechanism with different approaches is provided.

2.1 Smart City

Smart city is an urban area that produces and uses data from many sensors called Internet-of-Things (IoT) devices. These data collections help to improve life, health, comfort and resource management of the city. The difference between a regular city and a smart one is in IoT devices that collect all source of data and then these data collections are analyzed by artificial intelligence (AI) or pattern recognition algorithms. These results can late be used for above-mentioned improvements of cities. It is much more that one can imagine. One example that already has an impact in cities is a smart rubbish bin called Bigbelly[1]. It senses the amount of rubbish inside and it sends a request for collecting the rubbish if it is getting full. Moreover, it can serve as public Wi-Fi hot-spot or broadcast useful information via Bluetooth.

IoT devices in cities are connected to a network mainly via Wi-Fi, Bluetooth, LoRa, 6LoW-PAN, 4G or Ethernet technology. They usually do not have a lot of computing power. By its design, they try to be energy efficient because they are often battery powered. Therefore, these end-point devices are usually not suitable as peer-to-peer nodes. It is better if they can send data quickly and then sleep until the next time.

2.2 Peer-To-Peer

Peer-To-Peer (P2P) system is different from a client-server system. Devices connected in a P2P system behave as both, server and client at once. Therefore, every peer in the P2P system can provide some of its resources. Be it bandwidth, storage capacities, files or CPU/GPU cycles. Nodes, connected devices to the P2P system, are considered unreliable and often even untrusted. The P2P system is generally a virtual overlay network on top of the physical topology of the network in which the node is connected. The application layer peers are able to communicate directly via the logical overlay network. Moreover, this communication network can be without central control. Based on the architecture of the overlay network we distinguish between three types of P2P systems. The first is centralised, the second is decentralised and the third is hybrid.

[1] http://bigbelly.com/

Furthermore, based on the topology, a decentralised P2P system can be unstructured or structured [7]. See Figure 2.1.

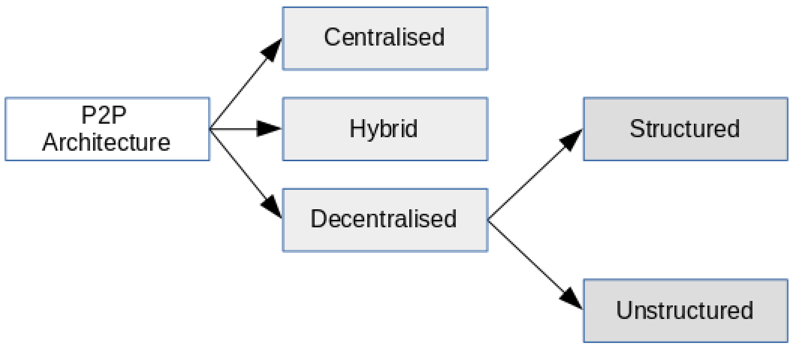

Figure 2.1: P2P Architecture

2.2.1 Centralised P2P system

Centralised P2P system combines both architecture patterns, client-server and P2P. A node first connects to the central server, often called a broker, in order to locate the desired resource on the P2P network or the server acts as a task scheduler that coordinates tasks among peers. The first example of such network is well known Napster [8]. It was originally founded as P2P music sharing service. A node connected to the Napster asked for a location of the desired resource and the server sends an address that has it. Unlike client-server architecture, once a node has the address it communicates directly with the peer that holds the required data. See Figure 2.2. Napster was shut down and later reopened as a legal[2] music streaming service. The second example is BOINC [9] or SETI@home [10]. In this case, the server acts as a task scheduler. Nodes fetch work units from the server directly. The advantages of this architecture good control over the network, cheap discovery of peers that have the required resource because the server holds the central index of all peers and their resources. On the other hand, the disadvantages are similar to client-server architecture. The server is a single point of failure. This type of P2P network does not scale well with a large number of nodes connected to the network. Finally, centralised P2P networks are not robust enough. A few examples of such P2P networks include Napster [8], BOINC [9] and SETI@home [10].

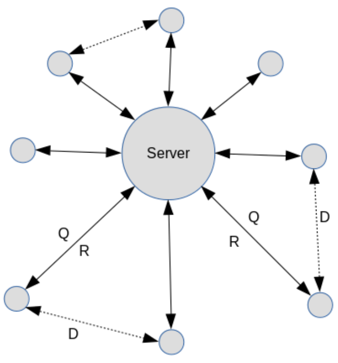

Figure 2.2: (Q) node queries the server. (R) response from the server. (D) data exchange.

[2]https://gb.napster.com/

2.2.2 Decentralised P2P system

Decentralised P2P system does not have any central index of resources and therefore no central point of failure. Every node in the decentralised P2P system has equal rights and responsibilities. A node has never complete knowledge about the network but only partial knowledge. If every node had complete information about all other peers in the network, every message would travel only one hop. On the other hand, every node would have to maintain a routing table of an O(N) size (where N is a number of nodes in a network). This would be unrealistic in P2P network of bigger size because each join and leave of a node would need to propagate to every peer in the network. The other extreme is if a node would have only two connections to each other in a ring topology. The cost of maintaining routing table would be minimal, but routing performance would be O(N). Decentralised P2P system radically improves robustness and scalability in comparison to a centralised P2P system. However, it introduces a new challenge of locating the desired resource (discovery service) on the P2P network. Two main approaches attempt to solve this issue. Based on the architecture of peers connecting in the overlay network, the approaches can be divided into structured and unstructured decentralised P2P systems. [11]

Structured overlay network maintains placement of resources or pointers to nodes that have desired resources under predefined rules. Generally, in structured P2P network this knowledge is maintained as distributed hash table (DHT[3]) [12]. A DHT provides lookup service for connected peers in a similar way as a standard hash table. A node is responsible only for a subset of key-value pairs of the DHT in such a way that nodes joining and leaving cause a minimal amount of disruption.

As it was described previously, the number of connections that a node have with its peers (degree of a node) influences the routing performance and structure of the overlay network. Since networks can be modelled as graphs, they can be studied and evaluated with the help of graph theory [13]. The table 2.1 and Table 2.2 show the specific properties of widely used structured P2P overlay networks. Therefore, It is important to choose the right one based on specific criteria.

Table 2.1: Asymptotic degree-diameter properties of the different graphs. (N is number of nodes in the graph)[13]

Graph	**Degree**	**Diameter D**
de Bruijin	k	$log_k N$
Trie	k+1	$2 log_k N$
Chord	$log_2 N$	$log_2 N$
CAN	2d	1/2 dN 1/d
Pastry	(b-1) $log_b N$	$log_b N$
Classic butterfly	k	2 $log_k N(1-o(1))$
Note: Degree is a number of connection every node has.		
Note: Diameter D is max distance (hops) that a message must travel.		

The next thing that has to be considered while choosing the graph structure is churn rate[4] on the P2P network. With every join and leave of a node, the routing table has to be updated. Usually, P2P networks such as Bittorrent[5] have high churn rate. Majority of nodes do not stay

[3] https://en.wikipedia.org/wiki/Distributed_hash_table
[4] https://en.wikipedia.org/wiki/Churn_rate
[5] http://www.bittorrent.com/

Table 2.2: Graph diameter for N = 10^6 (cells with a dash indicates that the graph does not support the corresponding node degree). [13]

k	de Bruijin	Trie	Chord	CAN	Pastry	Classic butterfly
2	20	-	-	huge	-	31
3	13	40	-	-	-	20
4	10	26	-	1,000	-	16
10	6	13	-	40	-	10
20	5	10	20	20	20	8
50	4	8	-	-	7	7
100	3	6	-	-	5	5

connected for more than 1 hour [14]. This introduces higher maintenance cost than in P2P network where nodes have incentives to be connected for a longer time period. One example can be Skype in early days where the median lifetime of a node was 5.5 hours [15]. As we can see churn rate is an important factor that needs to be considered while choosing the graph structure of P2P overlay network.

Another important thing affecting the performance of routing is locality. It is a relationship between overlay network and underlying physical network. If the physical network is not taken into account while constructing the overlay network, peers that are neighbours in the overlay network can be far away and a message has to do many hops in the physical network. Furthermore, peers that are on the same physical network can be very distant in the overlay network and a message has to do many hops in the overlay network. Consequently, this results in undesirable network traffic. [16]. This problem is illustrated in Figure 2.3.

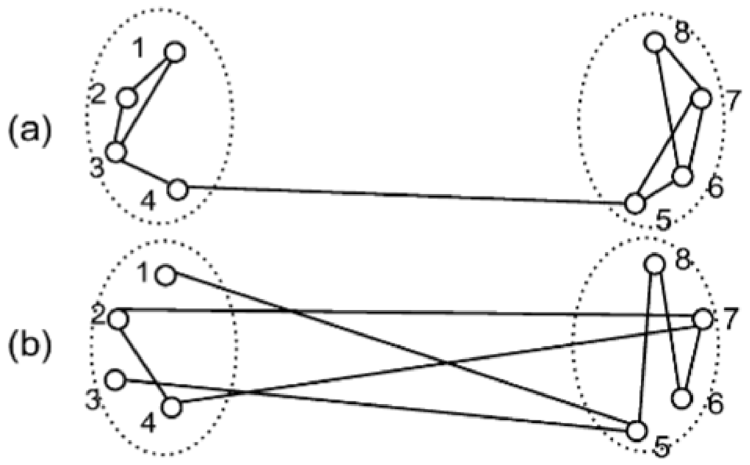

Figure 2.3: Illustration for locality-aware overlay and (b) randomly connected overlay [16]

There are many implementations of DHT that were developed. CAN [17], Chord [18], Pastry [19] and Tapestry [20] were the first one. Later, important improvements in terms of performance were made. CoralCDN[6] achieved the improvement through "a latency-optimized hierarchical indexing infrastructure based on a novel abstraction called a distributed sloppy hash table, or DSHT" [21]. DSHT helped also prevent hot-spot congestion (overloading a node when a specific

[6]http://www.coralcdn.org/

key becomes very popular). Moreover, Coral maintained a hierarchy of DSHT clusters based on region and size and therefore, allowed "nodes to locate nearby cached copies of web objects without querying more distant nodes" [21]. Also, important security improvements in DHT were made with the introduction of S/Kademlia [22] that has high resilience against common attacks. It uses cryptographic puzzles in order to limit free NodeId generation. S/Kademlia also uses parallel lookups over multiple disjoint paths over the network. Initial evaluation has shown that even with 20% of adversarial nodes in the network, there is still 99% chance of a successful lookup [22]. More comprehensive discussion about DHT and structured P2P is above the scope of this work. More information can be found in this book [12].

Unstructured overlay network utilizes different approach how queries are forwarded between peers in the overlay network. Every node maintains only its own data and keeps track of its connected neighbours that it can send queries to. However, this maintenance of a list of neighbours comes with a huge cost of bandwidth. Approximately 55% of all traffic is due to PING and PONG messages that serve for maintaining the list of neighbours [23]. As the name suggests there is no underlying structure that maps resources to nodes. Therefore, it is challenging to locate the desired resource because it is difficult to predict which node has it. Other difficulties are that there are no guarantees of completeness of answer (unless the whole network is searched) and response time [7]. Examples of such P2P networks are famous Gnutella[7] and FastTrack[8]. There are two main algorithms used for unstructured P2P networks. They are flooding and random walk.

The first routing strategy used in unstructured overlay P2P network is flooding. Consider an overlay network in Figure 2.4 where every node has degree between two and five (number of connected neighbours). With a higher degree, the distance between nodes reduces and a query has to do fewer hops in the overlay network. On the other hand, each node has to maintain a bigger list of its neighbours [11]. This list of neighbours can be shared between nodes. Once a node requires specific information it queries all its neighbours because it does not know the location of requested information. This process repeats further at each queried node until requested information is found or the maximum number of hops is reached. This limit for the maximum number of hops a message can do is called time-to-live (TTL). The TTL is a parameter of every message (query) and at each hop, it is decremented by one. It prevents queries circulates endlessly. Each node also keeps a list of queries that answered and if it receives the same query again it simply drops it [24].

The second approach used for routing queries in unstructured overlay P2P networks is random walk algorithm. It is very similar to flooding technique but it significantly decreases the communication cost. When a node issues or receives a query, it randomly selects neighbour (except the originator) and send the query further. However, the disadvantage is that the query processing time is very long. In order to improve the query processing time, the initiator could send k messages instead of only one [7]. See Figure 2.5.

2.2.3 Hybrid P2P system

Hybrid P2P system combines both, centralised and decentralised P2P systems. The main advantage of the centralised P2P system is fast lookup time but it has scalability issues. On the other hand, decentralised P2P system scale better but it requires longer time in resource locating.

[7]http://rfc-gnutella.sourceforge.net/
[8]https://en.wikipedia.org/wiki/FastTrack

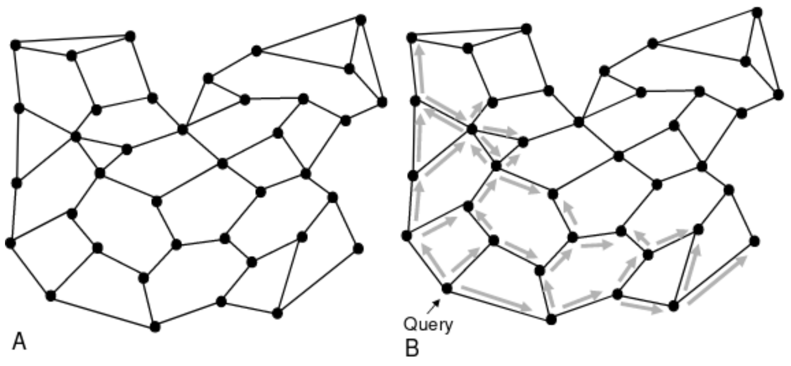

Figure 2.4: (A) Unstructured topology showing connections between peers and (B) query flooding to four hops. [11]

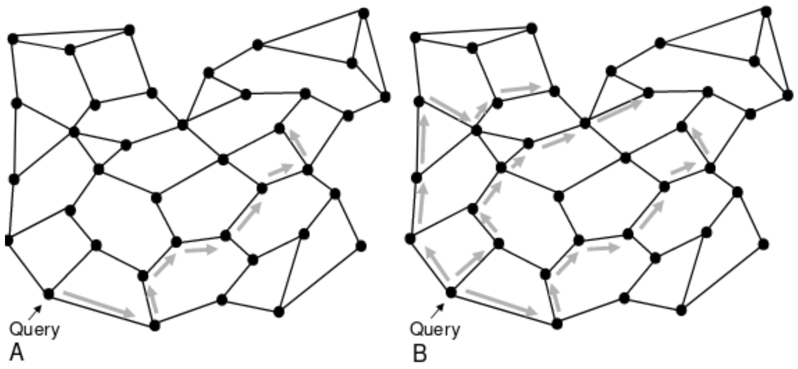

Figure 2.5: (A) Random walk and (B) k-way parallel random walk, k=3. [11]

In Hybrid systems a node can be selected as *super node* also known as *super peer* and serve other nodes as a server. There can be many criteria for selection of *super node*. Be it bandwidth, number of connections, longevity and many others. Therefore, resource locating can be done in centralised fashion (through supernodes) and also decentralised fashion [7]. It is clear that different P2P systems have their advantages and disadvantages. Therefore it is crucial to choose the right one or even combination of multiple approaches.

2.3 Distributed ledger

Distributed ledger technology (DLT) is a consensus of shared, synchronised and replicated records (financial or non-financial) spread across multiple geographic locations. There is no central data storage or single central authority. Users of DLT can use it to settle their transfers of data, money or assets without the need for trusted central authority. In traditional point of view, the central authority can be a financial institution such as a bank. DLT allows spreading the trust among participants instead of the traditional third party such as a bank. There are different types of DLTs. We can distinguish DLTs based on participation in the ledger, validation method and data structure of the shared records [25].

2.3.1 Permissioned VS Permissionless DLTs

There are two types of participation in the ledger. The first is unrestricted also known as permissionless. The second is restricted also known as permissioned[26]. In permissionless DLT anybody can participate in ledger update and validation. By definition, permissionless DLT is

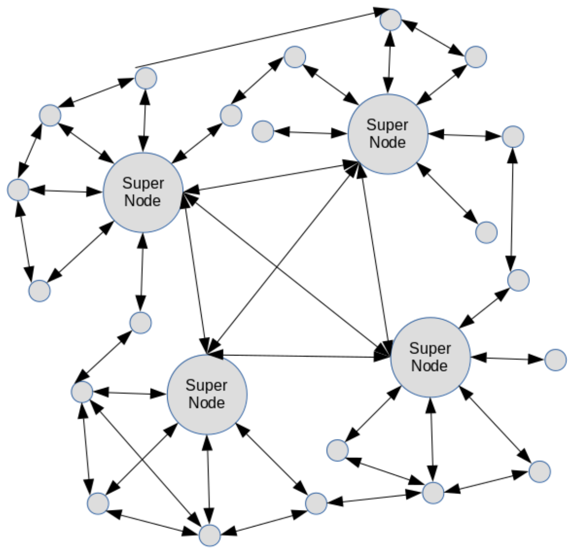

Figure 2.6: Hybrid P2P overlay network. Super nodes act as server for peers.

public. It means that the ledger is publicly available. On the other hand, in permissioned DLT only authorised participants can validate and update the ledger. Permissioned DLT can have private or public ledger [25].

The choice of permissionless or permissioned DLT can have an effect on maintenance costs as well as the range of possibilities to enforce truthful behaviour of validators. In permissioned DLTs, validators are known and they can be punished for malicious behaviour. It can be outside of the DLT (e.g. legal contracts, fines, etc.) as well as inside the DLT (e.g. disqualification from validation process, credibility, etc.). However, in permissionless DLTs validators are unknown and may be punished only inside the DLT. Therefore, game theoretic tools are used in permissionless DLTs as well as public-key cryptography [26]. Table 2.3 shows trade-offs between different types of DLT architectures. However, this table is only simplified version. While choosing the right DLT, it is crucial to get an excellent understanding of the specific DLT and its technical details.

Table 2.3: Trade-offs between different types of DLT architectures(simplified)

Properties of DLT	**Permissioned**	**Permissionless**
Speed	Faster	Slower
Energy efficiency	Better	Worst
Scalability	Better	Worst
Censorship resistance	No	Yes
Tamper - proof	No	Yes

2.3.2 Blockchain

In 2008 Satoshi Nakamoto published a white paper called "Bitcoin: A Peer-to-Peer Electronic Cash System" [27]. This paper revolutionized many industries. The idea of a cryptocurrency was not new but there was always a problem with double spend. Traditionally, this problem is solved via trusted third central authority such as bank instead of cryptographic proof. Satoshi Nakamoto did not invent new cryptographic methods, but he combines existing cryptographic methods, from 80's and 90's, in a new innovative way that allows the dawn of modern cryptocurrencies. This new invention is called a blockchain. Even though that in the original paper Satoshi Nakamoto did not mention the word blockchain, he describes the underlying principles of keeping transactions in *blocks* that are *chained* together via cryptographic hash (SHA256). He argued that the only way of preventing double spending is to know about all transactions. Even though that many people use terms blockchain and DLT interchangeably, it is considered to be only one type of DLT.

The fundamental principle of a generalised blockchain can be described as following. Records are grouped into blocks that one can imagine as a single page in a ledger. The next step is to create a cryptographic hash, such as SHA256, from this block and widely publishing it. The time when the hash is published is crucial. It proves that the records must have existed at the time to get into the hash. This is a timestamp of the block. Since each block contains a hash of the previous block, it forms a chain with each additional block reinforcing the ones before it. See Figure from the original Bitcoin paper [27].

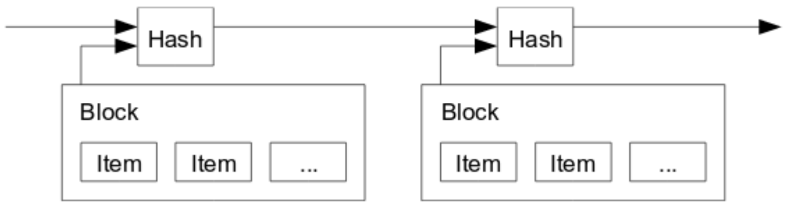

Figure 2.7: A chain of blocks with records [27].

In order to add a new block to the existing blockchain, the majority (in some cases at least 2/3) of the validators in the overlay P2P network have to agree on the newly created block. In permissioned blockchain, where we know the validators' identities and we can trust them, this proposal and validation of a new block is a straightforward process. On the other hand, in a permissionless blockchain where the validators are unknown and they can behave maliciously, we have to introduce countermeasures. An adversarial entity could create multiple nodes in the overlay P2P network and overvote the majority of honest nodes. This is known as Sybil attack[9]. In order to prevent this situation, Satoshi Nakamoto proposed a solution in a form of Proof-of-Work consensus algorithm. This is further discussed in Subsection 2.3.3

A public blockchain is by its definition traceable and linkable. Everybody can see and track every record, source and destination address from the first (genesis) block up to the latest one. The first cryptocurrency that implemented and used public blockchain is Bitcoin[10]. However, CriptoNote[11] introduced the idea of completely anonymous transactions on a public blockchain. This is achieved with help of multiple cryptographic methods such as ring signature and ringCT, stealth address, ITP router and Pedersen commitment [28]. Explanation of how this type of anonymous blockchain works is out of the scope of this work. Examples of such anonymous cryptocurrencies are Monero[12], AEON[13] and Bytecoin[14]. In these cryptocurrencies, it is very difficult (almost impossible) to find the address of the sender and receiver, see any transaction or even

[9] https://en.wikipedia.org/wiki/Sybil_attack
[10] https://bitcoin.org/
[11] https://cryptonote.org/
[12] https://getmonero.org/
[13] http://www.aeon.cash/
[14] https://bytecoin.org/

see how much money was sent. Despite this, it is possible to validate the transaction and prevent double spending.

2.3.3 Consensus Protocol

Consensus algorithm is essential for DLT. Since the ledger is distributed, it is crucial to reach consensus between nodes about every single transaction. This is specifically difficult in unreliable environments such as P2P networks are. In fact, it turned out to be impossible to reach consensus even with one faulty process in an asynchronous environment where no assumptions about message delivery delays or relative speeds of processes are made [29]. This proof is known as FLP impossibility. Therefore, all consensus protocols used in DLT are partially synchronous. Meaning that there are hard deadlines for validators.

In a case of permissioned DLT, any consensus protocol can be used to replicate the machine state. For example, a famous state machine replication protocol is Paxos[15]. This protocol is widely used in the industry for state machine replication. Its disadvantage is that it is difficult to understand and not Byzantine fault tolerant[16] (BFT) [30]. In 1999, Miguel Castro and Barbara Liskov published a paper "Practical Byzantine Fault Tolerance" (PBFT) that describes a new replication algorithm that is able to tolerate Byzantine faults [31]. This was a breakthrough because before there was not fast and efficient BFT protocol that could be used in real life. Recently a new effort has been made to improve the existing BFT state machine replication protocols. Tendermint[17] is one example that is already implemented as a pluggable consensus protocol for blockchains [30]. Another example of a blockchain that utilizes PBFT protocol is Hyperledger Fabric [32]. These examples consensus protocols solve only the issue of state machine replication. However, as it was mentioned previously, this would not be enough for permissionless DLTs due to the Sybil attack. In order to mitigate it, Satoshi Nakamoto proposed to use proof-of-work (PoW) for generating a new block on the blockchain [27].

Proof-of-work (PoW) is not a new idea. In 1992, Cynthia Dwork and Moni Naor invented the concept of "a computational technique for combatting junk mail in particular and controlling access to a shared resource in general" [33]. In short, the requester of a service first has to do some computational work in order to use the requested service. Essentially, it introduces a cost of accessing the requested service via economic measure because the user's hardware, time and electricity are not for free. In the case of permissionless DLT, a Proof-of-Work is used to prevent the Sybil attack and flooding of the P2P network with fake new blocks.

Bitcoin was the first cryptocurrency that used the idea of Proof-of-Work in the system. A node, in order to propose a new block, first have to solve a cryptographic puzzle. To be more specific, Bitcoin uses SHA256 cryptographic hash of a block (instead of a whole block it contains a Merkle tree root that is explained further). This hash of a newly proposed block must fulfil requirements of *difficulty* which is a specific number of leading zero bits in the hash. This is achieved by adding a nonce that can be altered into the input of the hash. See Figure 2.8 that shows how a previous hash, nonce and transactions creates a blockchain. With more zeros, the difficulty of a generating such a hash is exponentially increasing. The whole process is as follows.

[15]https://en.wikipedia.org/wiki/Paxos_(computer_science)
[16]https://en.wikipedia.org/wiki/Byzantine_fault_tolerance
[17]https://tendermint.com/

New transactions are broadcast to all nodes. Then each node collects new transactions into a block and it works on finding a difficult proof-of-work for its block. When a node finds it then it broadcasts the block with the hash to all nodes. Nodes should accept the block only if all the transactions are valid and they can easily verify that the work has been done by hashing the block and comparing those two hashes. Nodes express that they accept the new block by using the hash of the accepted block as a previous hash in a next block in the chain [27]. This is a simplified description of consensus protocol with proof-of-work that Bitcoin uses.

Permissionless DLT works with an assumption that majority of validators are hones. In the case of Bitcoin and similar DLTs that utilize proof-of-work in their consensus protocol, not the majority of validators but the majority of the CPU power have to be honest. The longest chain represents the majority because there was invested the most of the proof-of-work effort. Therefore, the longest chain will grow the fastest and outpace any competing chains. If an attacker wants to modify a past block, she would have to redo the proof-of-work of the specific block and all blocks after it and overtake the chain produced by honest nodes. Satoshi Nakamoto showed that the probability of a slower attacker catching up decreases exponentially as subsequent blocks are added [27]. Moreover, validators (miners) also have economic incentive, in form of reward, to continue the work and remain honest. Imagine that an attacker is in control of more computing power than all the honest nodes. She would have to decide between using the computation power to defraud people or earn new tokens (reward for creating new blocks and/or transaction fees). Remember that she would still have to spend a considerable amount of her own resources due to proof-of-work. By using it to steal back her payments, she would depreciate the market value of the tokens. Since this would be an undesirable situation for the attacker, she is disincentivised to do so.

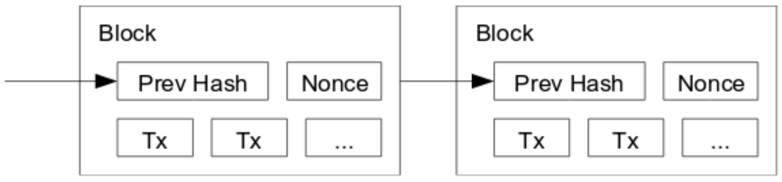

Figure 2.8: A chain of blocks with records that also contains a hash of the previous block and nonce [27].

Merkle tree is a cryptographic hash tree where every leaf node contains a hash of a data block and every node, that is not a leaf, contains a hash of its children's hashes [34]. Merkle tree is used to verify data that are transferred, stored or handled between computers or storages. It is often used in P2P networks (Gnutella[18]), file systems(Btrfs[19], IPFS[3], ZFS[20]) and DLTs (Bitcoin[27], Ethereum[21], etc.). The advantage of using Merkle tree, instead of hashing the whole file, is that single block of data can be verified faster without the need of having all blocks of the file. To be more specific, in the case of a blockchain, separate transactions are hashed. These hashes are leaves of the graph as shown in the Figure 2.9. Then it is not necessary to store all the transactions

[18]http://rfc-gnutella.sourceforge.net/
[19]https://btrfs.wiki.kernel.org/
[20]https://en.wikipedia.org/wiki/ZFS
[21]https://www.ethereum.org/

in a computer in order to verify a single transaction. This saves space and speed up verification process because the old blocks can be compacted by pruning the tree and leaving only the unspent transactions [27].

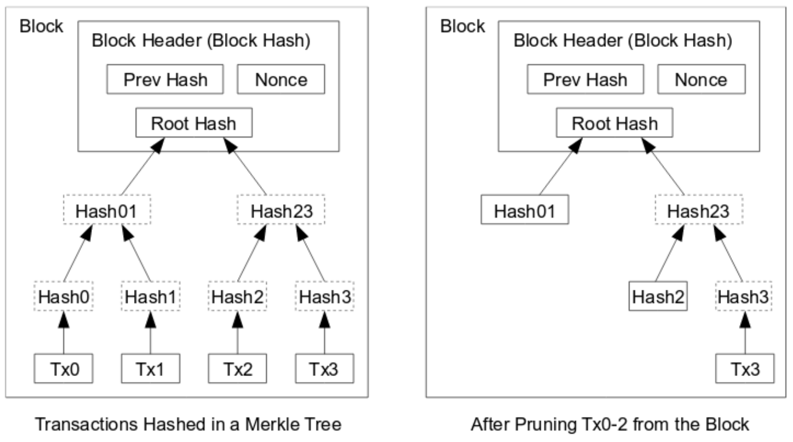

Figure 2.9: Transactions are hashed in a Merkle Tree and root hash is included in the block hash instead of the whole block. For verifying if Tx3 is part of the block, we need only root hash, hash01, hash2 and the Tx3 [27].

Proof-of-stake (PoS) is another category of consensus algorithms that is used in permissionless DLTs. The basic concept is that validators have to make a secure deposit (bond some stake) if they want to participate in consensus. The weight of a vote depends on the size of validator's deposit (stake). In contrast to proof-of-work, proof-of-stake algorithms have significant advantages in terms of security, decreased the risk of centralization and energy efficiency. However, they have a major disadvantage that is "nothing at stake" problem discussed in the next paragraph. The first cryptocurrency that used proof-of-stake (with the combination of PoW) is Peercoin [35]. Ethereum should also soon change its consensus algorithm to proof-of-stake. This new implementation is called Casper[22].

This consensus algorithms have many advantages but in many implementations, including Peercoin, validators can be only rewarded for producing new blocks but not penalized. This is causing the problem of nothing at stake. In the situation where multiple chains are competing, the validator's incentive is to vote for every chain at the same time. Imagine a situation with two competing chains where the validator can vote on a chain A and get reward P = 0.9 or on a chain B where the reward is P = 0.1 or on both at once if possible. This is shown in Figure 2.10. This results in a violation of safety and there is no incentive to converge into a single growing blockchain [36].

In comparison to proof-of-work, doing so would require splitting one's computing power in half. Therefore, this approach is not lucrative. It is shown in Figure 2.11. In this situation when the validator votes for both competing chains, the reward for voting on chain A and B is decreased by 50% and the combined reward is, therefore, smaller than voting only for original chain.

Nothing at stake can be prevented with a mechanism called *slasher* that was first proposed by Vitalik Buterin (Co-Founder of Ethereum) [37]. This mechanism penalizes validators that vote for multiple blocks simultaneously. When such situation happens then the validator's deposit is deduced appropriately. This mechanism is not simple to implement and it is still under heavy

[22]`https://github.com/ethereum/casper`

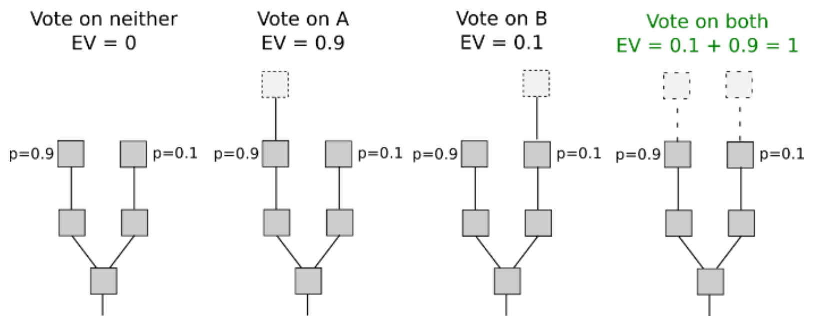

Figure 2.10: Nothing at stake problem. [36].

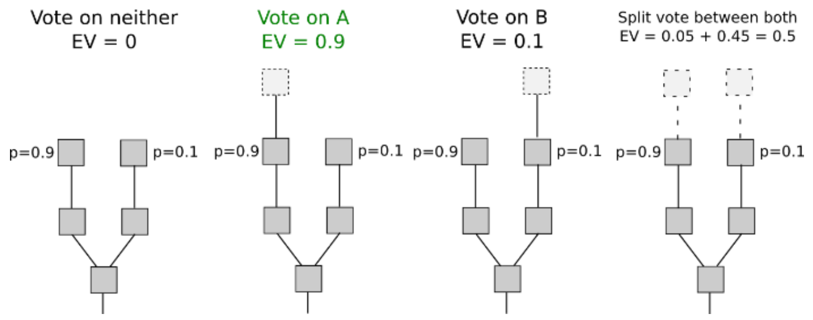

Figure 2.11: Proof-of-work, splitting one's computing power in half [36].

development for Ethereum Casper [36].

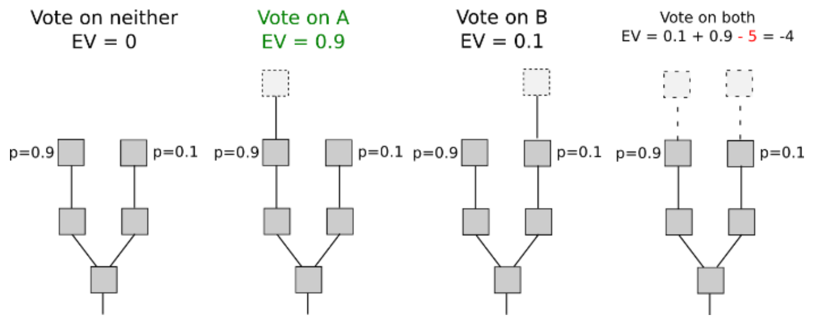

Figure 2.12: Nothing at stake problem with slasher [36].

Some people argue that even without *slasher* mechanism validators (stakeholders) have an incentive to act correctly and only vote for the original chain and not attacker's. The reason is that it is in their best interest to preserve the value of their investment because by supporting multiple chains at once the value of tokens on the market would decrease. On the other hand, this idea ignores that this incentive suffer from tragedy of commons[23] problems [36].

Above mentioned consensus protocols are the most used one in DLTs. However, consensus protocols in DLTs are currently important research topic and we can expect new inventions and breakthrough. There are also proposed and being implemented many other categories of consensus protocols such as proof-of-space, delegated proof-of-stake, proof-of-activity, proof-of-importance, proof-of-burn and many combinations of them.

[23]https://en.wikipedia.org/wiki/Tragedy_of_the_commons

2.3.4 Smart Contract

Smart contract was first proposed by Nick Szabo in 1995. The article "Smart Contracts: Building Blocks for Digital Markets" was published in magazine Extropy in 1996 [38]. He defined a contract to be "a set of promises agreed to in a meeting of the minds, is the traditional way to formalize a relationship." Contracts are mainly used in business relationships. Moreover, they are also used in personal relationships such as marriages, politics and other areas [38]. Szabo predicted in his article that digital revolution will dramatically change the traditional contracts. He called this new digital contracts "smart contracts".

The basic principle of a smart contract is that contractual arrangements between parties are written in a programming language. By storing this piece of code that defines the contractual arrangements into a blockchain, the smart contract becomes temper proof, self-executing and automatically enforceable. This reduces the need for the trusted third party and human intervention in the case of disagreements. Therefore the whole process of around traditional contracts is made less risky and more cost-effective. In order to create a smart contract, it must be able to be represented in logical flow such as "If X Then Y Else Z" [26]. Moreover, smart contracts must by deterministic otherwise nodes on the network would not reach consensus.

Ethereum was the first DLT that natively supported smart contracts. Ethereum provides a Turing complete virtual machine. It is called Ethereum Virtual Machine (EVM). Developers also created new programming language Solidity that is deterministic and suitable for smart contracts by its design [39]. Users can create their own contract, send them to Ethereum network where are replicated via BFT algorithm. For every execution of the contract a small fee, called Gas, has to be paid. Then the smart contract is sequentially executed on every node on the network.

Hyperledger Fabric is another example of a DLT that support smart contracts. In contrary to Ethereum, Fabric is permissioned blockchain. It rethinks the design and concepts used for permissionless DLTs and adapts them to suit better for permissioned DLT. It tackles existing limitations on permissionless DLTs such as execution throughput of smart contracts or the need for currency in public DLTs for smart contracts [40].

2.3.5 Distributed Ledger Technology in IoT & Smart Cities

With the rise in popularity of modern cryptocurrencies many researchers and companies started to explore the unexplored possibilities of DLT. During my research of related work I encountered numerous research papers and articles that made false claims or misunderstood the problematics. Often "Blockchain" technology is used in many proposed solutions without any proper explanation or logical reasons. However, In this section I will focus on the work I consider to be very useful and inspiring for this project.

In [41] authors did excellent summary of current blockchain technology and smart contracts. Moreover, they explored and discussed how smart contracts can be used in IoT and what should be considered for such deployment. One of the important things mentioned was that smart contracts are not legally binding (in permissionless DLT) and there is work being done towards solving this issue called "dual binding". On the other hand, I missed discussion about suitable consensus protocols for IoT devices. Despite it I consider it to be well written paper that served to me as introduction to this problematic.

In [42] an interesting concept was proposed to integrate Low-Power IoT devices to a

Blockchain. Since IoT devices are often powdered by battery and they have low computation power, they are not suitable for blockchain integration. Authors proposed decoupled model where IoT devices communicate with a gateway that is connected to a blockchain network and acts as a node. This gateway (a full blockchain node) can be queried remotely via a smart contract running on the blockchain. As a proof of concept they used private Ethereum network and decreased PoW difficulty in order to reach faster speed. The code is available on GitHub [24]. It is good example how low power IoT devices can be integrated to a blockchain. However, in a real world scenario, a private Ethereum network would not be the best choice. Since the network is private, there is no need for PoW consensus and existence of currencies. Therefore, more efficient solutions, such as Hyperledger Fabric [40] could be used. A discussion about this is missing in the paper.

One of the closest related work to this project is [43]. In this paper authors proposed "a decentralized access model for IoT data, using a network architecture that we call a modular consortium architecture for IoT and blockchains." With this architecture, they aim to provide IoT data privacy via blockchains and address the challenges associated with implementing blockchains to IoT. A single blockchain that would be responsible for logging every IoT data operation would not scale well. Therefore, they broke down the network into smaller private network called *sidechains*. These sidechains are interconnected via main *consortium* network. A consortium network runs own blockchain. It is responsible for access control and prevents any unauthorized access from one sidechain to another member of the consortium network. A sidechain is responsible for creation of legitimate IoT records and access control of incoming request transactions. These records are further stored in fully decentralised content addressed file system IPFS [3]. A hash of this record is updated in the smart contract that is responsible for the data transaction. Moreover, the same decoupled pattern of IoT devices and blockchain is presented. In this scenario IoT devices hold encrypted communication with the gateway (called "validator node") that is running the smart contract. However, authors proposed this solution where "The private IoT network consists of IoT devices and one validator node running the sidechain". There is no benefit or reason of running the sidechain on a single computer. Either, there should be multiple nodes on the sidechain for decentralisation (increased reliability) or there is no need for the sidechain. Also there is no implementation of the proposed solution, but only network traffic and processing overhead of existing candidate blockchain solutions. The evaluated blockchains were Monax and Ethereum.

Similarly in [44], authors proposed a solution where single blockchain is responsible for access control that is decoupled from storage layer. They consider IoT data to be streams and therefore introduced interesting concept of storing them in a specific data structure. Here are the data chunked, compressed and encrypted. Furthermore, these chunks are cryptographically chained together. This preserves the time-line of produced data. They also advocate distributed data storage for saving these data streams. They relay on P2P overlay network and DHT that serves as general-purpose private key-value data store interface. This paper presents good solution to IoT secure data storage with promising initial evaluation results. However, I think that this proposed solution would not scale well due to single blockchain. Moreover, it does not integrate payments for shared (sold) IoT data.

There are also other papers that are related to this work and discusses IoT and blockchain

[24]https://github.com/kozyilmaz/blocky

integration in smart cities. CitySense: blockchain-oriented smart cities[45]

Mind my value: a decentralized infrastructure for fair and trusted IoT data trading[46]

Towards an Optimized BlockChain for IoT[47]

Slock.it[25] Filament[26] chain of things[27]

[25]https://slock.it/
[26]https://filament.com/
[27]https://www.chainofthings.com/

Chapter 3

Requirements & Architecture

The application's functional and nonfunctional requirements are specified and justified in this chapter. Both of them are sorted based on priority. The requirements with the highest priority are listed first. Moreover, the system architecture is described and justified against requirements.

3.1 Functional Requirements

The goal of this project is to provide decentralised, secure, robust and physical distance aware solution for IoT devices in an interplanetary smart city. The solution has to provide easy means for trade with the data collection among untrusted parties. To achieve this goal, I need functional requirements from FR1 to FR5.

- **FR1 The solution will be fully decentralised peer-to-peer network with different options of connectivity, including Internet, Wi-Fi, Ethernet, etc.**

  In order to create a solution that does not have a central point of failure, it has to utilize a decentralised peer-to-peer network. Furthermore, this addresses the goal for robustness and physical distance awareness. The solution should not depend on specific connectivity technology because it would introduce limitation for future interplanetary cities.

- **FR2 The nodes should be able to exchange data without a centralised authority in a safe, secure and confidential manner and in an accountable fashion.**

  Since IoT data can consist of sensitive data or simply the data hold a certain value, the parties must be able to exchange the data in a safe, secure and confidential manner. Accountability of the mechanism will be a crucial part in the case of a dispute about a transaction among parties. These things should be achieved with the trust in cryptography rather than traditional third party. This requirement is also necessary to achieve the goal of providing easy means for trade with data collections.

- **FR3 The solution should record every transaction between nodes.**

  In order to achieve data exchange in an accountable fashion. The solution has to record every transaction as a tamper-proof log. This log will serve as a proof for involved parties. This requirement is necessary in order to provide easy means for trade among untrusted parties.

- **FR4 There will be provisions for two nodes to pair up adequately based on multiple parameters. For example freshness of the data, bandwidth, latency, etc.**

Multiple data parameters can play a vital role in the decision which provider (node) serves better, more suitable, data for a specific use. In some cases, the frequency of data is more critical then accuracy, in others reliability plays the most crucial role. Therefore, the solution has to provide the possibility to pair nodes adequately.

- **FR5 The node should be able to publish what data it can provide.**

  To provide an environment for data exchange, there has to be a way to get a list of available data and their parameters.

Due to the nature of this research topic, the above functional requirements (FR1 - FR5) are sufficient for reaching the stated goal. The reason is that this is an open-ended cutting-edge research topic and very detailed functional requirements would limit exploration possibilities.

3.2 Non-Functional Requirements

I list and justify non-functional requirements (NFR1-3) here:

- **NFR1 The solution will be scalable.**

  The environment and size of interplanetary smart cities can vary. For this reason, it is essential that the solution can cope with it. Moreover, the solution should be suitable for use in smart houses and interplanetary cities. Therefore, the solution has to be scalable.

- **NFR2 The solution will be robust. To be more specific, it will not fail as a result of individual components failing.**

  To address the problem of central point of failure, the solution has to be able to continue working even if individual components fail. This improves the robustness of the whole system.

- **NFR3 The solution will be resistant to a number of attacks. For example Sybil, Churn and DoS attack.**

  In order to address the possible threats in smart cities, the solution must be resistant to multiple attacks that are common in P2P networks and IoT infrastructure.

- **NFR4 All the technologies that will be used in this project should be free of charge.**

  All the technologies and tools should be free of charge. They do not have to be necessary open source.

3.3 System Architecture

The following section describes high-level system architecture that is designed to fulfil above mentioned functional and non-functional requirements. The architecture diagram 3.1 aims to explain the underlying structure of the system and provide an explanation of data flow.

Due to the importance of security and resistance against several attacks. I decided that the used DLT has to be private and not public. This brings also improvement in performance because there is no need for proof-of-work algorithm.

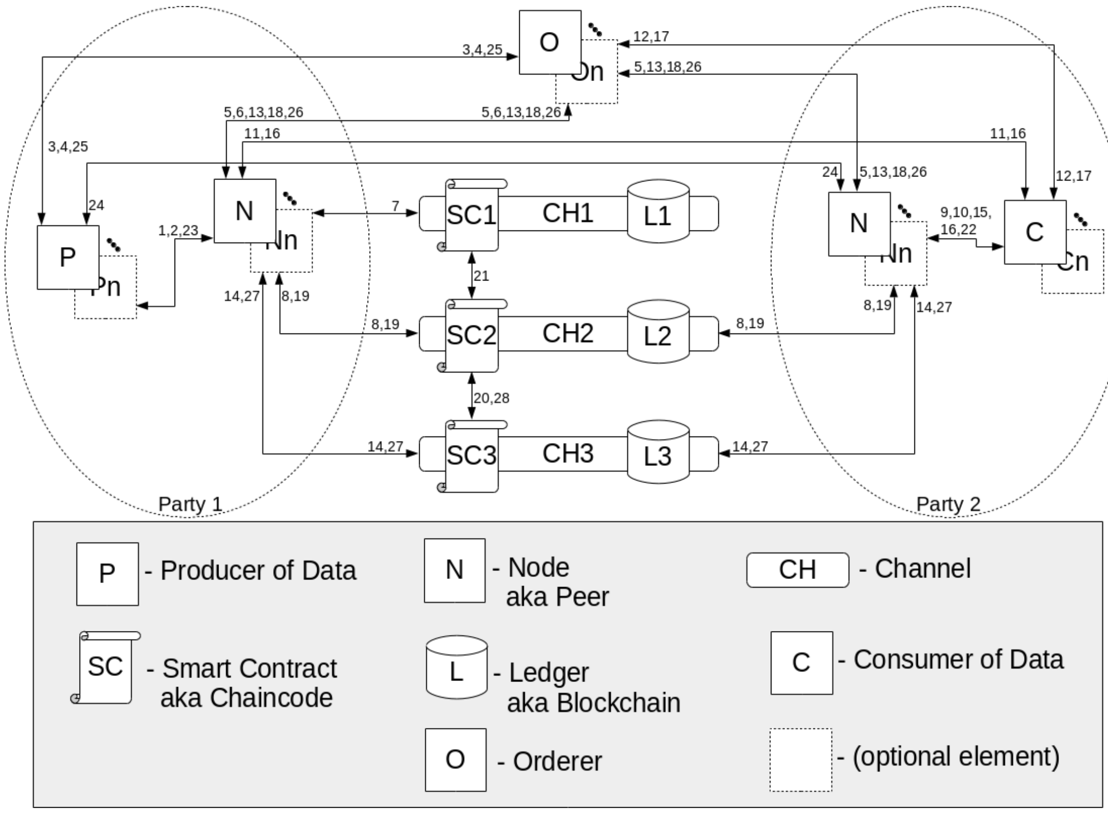

Figure 3.1: System Architecture

3.3.1 Producer of Data

The producer of data is an application that is responsible for capturing data from one or multiple sensors and IoT devices. Moreover, it communicates with one or multiple nodes and orderers in order to query or update the state of the ledger. In the scenario that involves payments for data, it may be also responsible for releasing tokens that were temporarily blocked until the transaction of data entry is finished. This process is further described in 3.3.8. This component is required in order to fulfil **FR2**, but it is not the only one.

3.3.2 Consumer of Data

Similarly, the consumer of data is an application that communicates with one or multiple nodes and orderers in order to update the state of the ledger. On the other hand, the consumer of data does not publish any data entry but it can request and pay for them. An application can be both, consumer and publisher of data entries at the same time. This component is required in order to fulfil **FR2**, but it is not the only one.

3.3.3 Ledger

A ledger, aka blockchain, immutably records all the transactions generated by smart contracts. It is hosted by a single or multiple nodes. Therefore it is distributed across them. Nodes are responsible for maintaining consistency of its state. This component is crucial for storing data and recording all transactions as a temper-proof log in order to satisfy **FR3**. In addition, it is partially required to fulfil **FR5** because without the storage a node could not publish anything.

3.3.4 Smart Contract

A smart contract, aka chaincode, is a piece of code which copy is held and executed by a node. This program execution can query or generate an update proposal for the ledger that is maintained by the node. A smart contract is invoked by an application (publisher or consumer). This component is required in order to fulfil **FR2**. However, this is only one component from multiple that are necessary to do this. Moreover, this component also satisfies the **FR4** because a smart contract can be designed such that it provides functions and logic for two nodes to pair up adequately based on multiple parameters.

3.3.5 Node

A node, aka peer, is a fundamental building block of the network because it hosts one or multiple smart contracts and ledgers. A node is responsible for maintaining consistency of ledgers and executing smart contracts. A node can have multiple ledgers and contracts. Furthermore, in comparison to orderers, it specialises in verifications of transactions. If an application wants to access a ledger, it has to do it via the node that maintains the ledger and has installed a copy of a smart contract. Because overall design is a fully decentralised peer-to-peer network that uses multiple nodes, this satisfies the **FR1**. Furthermore, this component is also necessary to satisfy **NFR2** and **NFR1**.

3.3.6 Orderer

An orderer is a specialised type of a node. In contrast to a regular node, its role is to collect transaction proposals from applications (publisher or consumer), create and distribute new blocks to all nodes that maintain the state of the specific ledger. There can be single or multiple orderers. In the case of multiple orderers, they use a consensus protocol to agree on the strict order and results of transactions. The consensus protocol is not hard-coded, but it is a pluggable module. Therefore, this can also support BFT (Byzantine Fault Tolerant) consensus protocol. This component satisfies the **FR1**, **NFR1** and **NFR2**. Since the roles of a regular node and orderer are decoupled, this design should provide a scalable decentralised solution.

3.3.7 Channel

A channel is a mechanism by which a set of nodes, orderers and applications within a blockchain network can communicate and transact privately. By joining a channel the mentioned components agree to collectively share and manage identical copies of the ledger or multiple ledgers for the specific channel. This module is required in order to satisfy **FR2**, **FR5** and **NFR3**. It is worth noting that even though the figure 3.1 shows SC and L as if it was in the channel, it simply means that every component that joins the channel shares and manages an identical copy of the ledger.

3.3.8 Data Flow

A scenario that describes data production, advertisement, purchase and retrieval are described in this section. The complete data flow of this scenario is graphically shown on the architecture diagram 3.1.

1. P creates data and contacts N (Party 1) using SC1 to generate a transaction proposal response for L1 on CH1 for new data entry. N (Party 1) responds to P with a signed endorsement for ledger update (new entry).

2. P contacts N (Party 1) using SC2 to generate a transaction proposal response for L2 on CH2 for new data entry advertisement. N (Party 1) responds to P with a signed endorsement for ledger update (new entry).

3. P sends the transaction to update L1 on CH1 with signed and endorsed the proposal from node N (Party 1) to O.

4. P sends the transaction to update L2 on CH2 with signed and endorsed the proposal from node N (Party 1) to O.

5. O creates a new block for L2 on CH2 and sends it to N (Party 1) and N (Party 2). L2 contains data entry advertisement (without values) produced by P.

6. O creates a new block for L1 on CH1 and sends it to only N (Party 1). L1 contains data entry (with values) produced by P.

7. N (Party 1) validates new block received from O and updates L1 on CH1.

8. N (Party 1) and N (Party 2) validates new block received from O and updates their copy of L2 on CH2.

9. C is notified by N (Party 2) that a new data entry advertisement was created in L2 on CH2 and decides that it wants to purchase it.

10. C contacts N (Party 2) to generate a transaction proposal response for L3 on CH3 to transfer tokens from C's account to P's account.

11. C also contacts N (Party 1) to generate a transaction proposal response for L3 on CH3 to transfer tokens from C's account to P's account. C must have valid, signed endorsement from both N (Party 1) and N (Party 2).

12. C sends the transaction to update L3 on CH3 with signed and endorsed proposals from node N (Party 1) and N (Party 2) to O.

13. O creates a new block for L3 on CH3 and sends it to N (Party 1) and N (Party 2). L3 contains account with tokens that are used for payments.

14. N (Party 1) and N (Party 2) validates new block received from O and updates their copy of L3 on CH3. Tokens that are transferred from C's account to P's account are marked as on hold because data entry value was still not revealed in L2 on CH2. Therefore, P cannot use the received tokens yet.

15. C contacts N (Party 2), with valid token transaction ID, to generate a transaction proposal response for L2 on CH2 to reveal data entry value for the advertisement.

16. C also contacts N (Party 1), with valid token transaction ID, to generate a transaction proposal response for L2 on CH2 to reveal data entry value for the advertisement. C must have valid, signed endorsement from both N (Party 1) and N (Party 2).

17. C sends the transaction to update L2 on CH2 with signed and endorsed proposals from node N (Party 1) and N (Party 2) to O.

18. O creates a new block for L2 on CH2 and sends it to N (Party 1) and N (Party 2).

19. N (Party 1) and N (Party 2) validates new block received from O and updates their copy of L2 on CH2. In order to finish this step, the validating nodes have to execute a special function in SC2 that invokes inter-channel communication described in next two steps.

20. SC2 query L3 via SC3 if the provided token transaction ID is already spent or still on hold. If it is still on hold (unspent for data purchase) then the flows continue.

21. SC2 query L1 via SC1 if the data entry ID is present in L1. If it is then the flow continues, data entry value is retrieved from L1 and updated into L2.

22. C receives revealed data entry value that paid for.

23. P contacts N (Party 1) using SC3 to generate a transaction proposal response for L3 on CH3 to release the tokens used for the revealed data entry value.

24. P also contacts N (Party 2) using SC3 to generate a transaction proposal response for L3 on CH3 to release the tokens used for the revealed data entry value. C must have valid, signed endorsement from both N (Party 1) and N (Party 2).

25. P sends the transaction to update L3 on CH3 with signed and endorsed the proposal from node N (Party 1) and N (Party 2) to O.

26. O creates a new block for L3 on CH3 and sends it to N (Party 1) and N (Party 2).

27. N (Party 1) and N (Party 2) validates new block received from O and updates their copy of L3 on CH3. In order to finish this step, the validating nodes have to execute special function in SC3 that invokes inter-channel communication described in next steps.

28. SC3 query L2 via SC2 if the data entry value was revealed. If it is true then the tokens are released.

3.3.9 Summary

This section described and introduced the system architecture that should fulfil all the functional and non-functional requirements. Furthermore, the figure 3.1 completely describes data flow in a typical scenario.

Chapter 4

Methodology, Technologies & Implementation

The methodology, technology, development tools and project risk assessment are ~~further~~ described in this chapter. In addition, I also explain the major reasons behind the decisions.

4.1 Methodology

This project aims to explore the possibilities of utilizing blockchain technology, the best concepts from multiple peer-to-peer protocols and ideology of IPFS [3] for interplanetary smart cities. This topic currently gained a lot of interest and became an active field of research. As we could read in the section Blockchain in IoT & Smart Cities, there are many different approaches to address this problematic. Therefore, it is open-ended research topic and the traditional software development techniques would not be suitable. Hence, I decided to use exploratory programming technique. These steps were followed in order to achieve progress and develop the application.

- Background literature reading and understanding the research topic.
- Literature review of existing similar research projects.
- Developing a project plan
- Learning about relevant software technologies, programming languages and tools used in this research area.
- Creating functional and non-functional requirements for the application.
- Designing the system architecture that can solve the stated challenges.
- Experimenting and exploring many potential technologies, languages and tools that could be suitable for this application.
- Implementing the prototype.
- Developing the system and implementing functionalities needed to fulfil all requirements.
- Evaluating and testing the developed system.
- Writing this report and user manual.
- Summarizing the project in conclusion and discussing future work.

4.2 Technology

This section describes the technology and tools ~~that were~~ used for the development of the system. There are also described reasons that were behind the decisions for specific technology and toolset.

4.2.1 Distributed Ledger Technology

In terms of DLTs, I considered several popular options and assessed if they are suitable for the task. Since I knew that smart contracts are crucial for the whole concept, I immediately focused on DLTs that support them. The final candidates were Ethereum[1], EOS[2], and Hyperledger Fabric[3]. Despite that Ethereum uses proof-of-work algorithm, it is possible to change it for a different one. One example can be Tendermint[4] that is BFT (Byzantine Fault Tolerant) consensus algorithm for blockchains mentioned in 2.3.3. However, this approach would require a significant amount of work and time. Moreover, there would still be the need for coins (gas) to use smart contracts. The result would be uncertain and the main research focus could be unexplored.

The second DLT is EOS that appeared to be an ideal candidate. It might be even more suitable than the chosen one because it is specially designed for the high performance of smart contracts from the beginning. However, It is still under a heavy development and in alpha phase during the time of writing this report. Therefore, I decided not to use it.

The last DLT is Hyperledger Fabric. It has a big community support and a stable release version (v1.0 at the time of writing). In addition, there is no need for cryptocurrency if smart contracts are used because it is designed as a private DLT. Furthermore, it was developed, since the beginning, for business purposes in mind. It supports concurrent execution of smart contracts and a node can have multiple smart contracts and ledgers that are separated by secure channels. For these reasons, I decided to use Hyperledger Fabric.

4.2.2 Scripting and Programming Language

Choosing the right programming language for a project is very important. This decision can improve readability, performance, future maintainability and decrease development time. For these reasons I carefully considered several languages that can be used for writing smart contracts for Hyperledger Fabric and also programming languages that can be used for interaction with node's API.

For writing smart contracts I could use either Java or Go language. The native language of Hyperledger Fabric is Go and the majority of smart contract examples are written in this language. Even though I already knew Java, I realised that it would be better if the smart contracts are written in Go. If I had some problems I could ask in the community that always provided examples in Go rather than in Java. Moreover, if I used only Go and no Java there is no need to install JVM that can be convenient for some devices running this system. For these reasons, I decided to learn a completely new language that I did not know before. It turns out that it was a great choice.

For interacting with the smart contract API I could choose from four languages. These are Java, Node.js, Go and GNU Bash. Again I carefully considered the options and tried each of them.

[1]https://github.com/ethereum/go-ethereum
[2]https://github.com/EOSIO/eos
[3]https://github.com/hyperledger/fabric
[4]https://tendermint.com/

Because the main focus of this project is not a development of a client application (publisher or subscriber) but the underlying mechanism, I decided to use Bash scripting language that interacts via CLI (command line interface). This allowed me to quickly adjust developed scripts during the phase of fast prototyping of smart contracts. Furthermore, any other language can be used for the development of a client application in the future.

4.2.3 IDE and Compiler

For the development of smart contracts, I decided to use Microsoft Visual Studio Code because I am familiar with it and using it every day. In addition, it has a good integration with Git version control that I also used in the development process. It also provides a convenient way for debugging because it integrates DBG debugger.

I considered two most known compilers. The first one is original native "go tool compile" also known as "gc" and the second one is "gccgo"[5]. The main difference is that gccgo compiles slower but can increase performance for specific CPU architecture. In this project, the more important factor for the development is fast compile time because after every change, the source code has to be recompiled. This allowed faster prototyping of smart contracts.

4.2.4 Source Control

As a source control, I decided to use Git version control because it is stable an widely used in the industry. Moreover, I am familiar with Git as a daily user. For the development of the smart contracts and bash scripts, a single master branch was set up. This repository was after every significant change backed up to a remote location on Github.com.

4.2.5 Containers

For better scalability and encapsulation Hyperledger Fabric uses Docker[6] containers. Every component such as a node, orderer and even smart contract is running in a separate environment inside a Docker container. If there is a need to use more nodes or orderers to scale up, we can just run new instances. It is very simple but powerful concept that is widely used in the industry in recent years.

4.2.6 Storage for The Ledger

There were two options that could be used as a storage for the ledger. The first one was CouchDB[7] and the second was LevelDB[8]. The advantage of CouchDB is that it enables rich query against the data when values are modelled data, as JSON data. Even though it seemed to be a better choice than LevelDB, during the evaluation I found out that it posed a limitation for transaction throughput in my set up. Simply, some transactions were not committed to the new state. Therefore, I had to switch to LevelDB that turned out to be better suited for the developed system.

4.2.7 Summary

This section discusses the chosen technology and tools for the development of the system. Among many DLTs, I decided to use Hyperledger Fabric v1.0. As a programming language for

[5]`https://blog.golang.org/gccgo-in-gcc-471`
[6]`https://www.docker.com/`
[7]`https://couchdb.apache.org/`
[8]`https://github.com/google/leveldb`

smart contract I used Go language. GNU Bash scripts are used for interfacing with a node and an orderer. All of the code base was written in Microsoft Visual Studio Code. As a version control I used Git and every component is containerised with Docker. As a storage for the ledger I firstly used CouchDB, but later it was swapped for LevelDB that turned out to be more suitable.

4.3 Implementation

This section describes the process of the implementation. At the beginning I started to implement the most important smart contract for storing and retrieving data from the ledger. The second smart contract that was developed provide basic functionality for accounts and payments between parties. This is necessary only if the data exchange should be paid. After developing these two smart contracts I could continue with the third one that interacts with both of them. The third smart contract provides functionality for publishing an advertisement for data entry as described in figure 3.1. These three smart contracts provide an API for the whole system.

4.3.1 Throughput

After implementing all three smart contracts I did initial evaluation and explored major drawbacks of the implementation. More details are provided in the following Chapter 5. Therefore, I decided to completely rewrite all three smart contracts so that they are now able to handle high throughput. It is possible to look at and run the first implementation too. This is possible with simple command "$ git checkout 8cb9365c5e4070568fbb6112c6c5eb1b80ab655b" and to get back to latest commit use command "$ git checkout master"

4.3.2 Smart Contract for Data Entry

The main tasks of this smart contract are quite simple. There have to be functions that create data and retrieve data from the state of the ledger dedicated for this purpose. As a first thing, I had to design a structure for the data that should be stored in the ledger.

```
type DataEntry struct {
    RecordType   string // RecordType is used to distinguish the various
                        // types of objects in state database
    DataEntryID  string // unique compound key that is ID~CreationTime
    Description  string // human readable description
    Value        string // data value
    Unit         string // optional units for the data value
    CreationTime uint64 // Time when the data was created.
                        // It can differ from the blockchain entry time
    Publisher    string // publisher of the data
}
```

Every variable except CreationTime is string data type. This was done deliberately because I did not want to introduce any limitation. For example DataEntryID could be hash string or Value could be in hexadecimal etc. On the other hand, Creation time i unsigned integer for a specific reason. I wanted to provide a function that can return the latest entry and this is done by comparison operator.

Functions that can be invoked in this smart contract are:

1. Create data
2. Get data by its ID
3. Get all data by its ID
4. Get latest data by its ID
5. Get data by publisher

4.3.3 Smart Contract for Tokens

```
// Account represents account of a member
type Account struct {
    RecordType string // RecordType is used to distinguish the various
                      // types of objects in state database
    AccountID  string // unique id of the account
    Name       string // name of the account holder
    Tokens     int64  // amount of tokens (money)
}
```

4.3.4 Smart Contract for Data Entry Advertisement

```
type DataEntryAd struct {
    DataEntry        // anonymous field
    Price     int64  // Price for data value
    AccountNo string // account number where to transfer tokens
}
```

4.3.5 Inter Channel Communication

Chapter 5

Testing & Evaluation

5.1 Testing

5.2 Evaluation

Chapter 6

Discussion, Conclusions & Future Work

6.1 Discussion

6.2 Conclusions

6.3 Future Work

Bibliography

[1] "World's population increasingly urban with more than half living in urban areas | UN DESA | United Nations Department of Economic and Social Affairs," Jul. 2014. [Online]. Available: https://www.un.org/development/desa/en/news/population/world-urbanization-prospects.html

[2] "WHO | 7 million premature deaths annually linked to air pollution," Mar. 2014. [Online]. Available: http://www.who.int/mediacentre/news/releases/2014/air-pollution/en/

[3] P. Labs, "IPFS is the Distributed Web." [Online]. Available: https://ipfs.io/

[4] S. Curtis, "How much is your personal data worth?" Nov. 2015. [Online]. Available: http://www.telegraph.co.uk/technology/news/12012191/How-much-is-your-personal-data-worth.html

[5] "Logitech Will Intentionally Brick All Harmony Link Devices Next Year." [Online]. Available: https://www.bleepingcomputer.com/news/hardware/logitech-will-intentionally-brick-all-harmony-link-devices-next-year/

[6] S. Duzellier, "Radiation effects on electronic devices in space," *Aerospace Science and Technology*, vol. 9, no. 1, pp. 93–99, Jan. 2005. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S1270963804001129

[7] Q. H. Vu, M. Lupu, and B. C. Ooi, *Peer-to-Peer Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. [Online]. Available: http://link.springer.com/10.1007/978-3-642-03514-2

[8] "Napster," Jan. 2018, page Version ID: 820643659. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Napster&oldid=820643659

[9] "BoincPapers âĂŞ BOINC." [Online]. Available: https://boinc.berkeley.edu/trac/wiki/BoincPapers

[10] "SETI@home." [Online]. Available: https://setiathome.ssl.berkeley.edu/

[11] J. F. Koegel Buford, H. H. Yu, and E. K. Lua, *P2P networking and applications*, ser. The Morgan Kaufmann series in networking. Amsterdam ; Boston: Elsevier/Morgan Kaufmann, 2009, oCLC: ocn267167232.

[12] D. Korzun and A. Gurtov, *Structured Peer-to-Peer Systems*. New York, NY: Springer New York, 2013. [Online]. Available: http://link.springer.com/10.1007/978-1-4614-5483-0

[13] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh, "Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM, 2003, pp. 395–406.

[14] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *Proceedings*

of the 6th ACM SIGCOMM conference on Internet measurement. ACM, 2006, pp. 189–202.

[15] S. Guha and N. Daswani, “An experimental study of the skype peer-to-peer voip system,” Cornell University, Tech. Rep., 2005.

[16] X. Zhang, Q. Zhang, Z. Zhang, G. Song, and W. Zhu, “A Construction of Locality-Aware Overlay Network: mOverlay and Its Performance,” *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 18–28, Jan. 2004. [Online]. Available: http://ieeexplore.ieee.org/document/1258112/

[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, *A scalable content-addressable network*. ACM, 2001, vol. 31, no. 4.

[18] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan, “Chord: a scalable peer-to-peer lookup protocol for internet applications,” *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17–32, Feb. 2003. [Online]. Available: http://ieeexplore.ieee.org/document/1180543/

[19] A. Rowstron and P. Druschel, “Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems,” in *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer, 2001, pp. 329–350.

[20] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, “Tapestry: a resilient global-scale overlay for service deployment,” *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 41–53, Jan. 2004.

[21] M. J. Freedman, E. Freudenthal, and D. Mazieres, “Democratizing Content Publication with Coral.” in *NSDI*, vol. 4, 2004, pp. 18–18.

[22] I. Baumgart and S. Mies, “S/kademlia: A practicable approach towards secure key-based routing,” in *Parallel and Distributed Systems, 2007 International Conference on*. IEEE, 2007, pp. 1–8.

[23] M. Ripeanu, I. Foster, and A. Iamnitchi, “Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design,” *arXiv preprint cs/0209028*, 2002.

[24] I. J. Taylor and A. B. Harrison, *From P2P and grids to services on the web: evolving distributed communities*, 2nd ed., ser. Computer communications and networks. London: Springer, 2009, oCLC: 254593378.

[25] A. Pinna and W. Ruttenberg, “Distributed Ledger Technologies in Securities Post-Trading Revolution or Evolution?” 2016.

[26] J. Mattila, *The blockchain phenomenon*. Berkeley Roundtable of the International Economy, 2016. [Online]. Available: https://www.researchgate.net/profile/Juri_Mattila/publication/313477689_The_Blockchain_Phenomenon_-_The_Disruptive_Potential_of_Distributed_Consensus_Architectures/links/589c31caa6fdcc754174493a/The-Blockchain-Phenomenon-The-Disruptive-Potential-of-Distributed-Consensus-Architectures.pdf

[27] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 2008.

[28] S. Noether, “Ring Signature Confidential Transactions for Monero,” Tech. Rep. 1098, 2015. [Online]. Available: http://eprint.iacr.org/2015/1098

[29] M. J. Fischer, N. A. Lynch, and M. S. Paterson, “Impossibility of distributed consensus with

one faulty process," *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 374–382, 1985.

[30] E. Buchman, "Tendermint: Byzantine Fault Tolerance in the Age of Blockchains," PhD Thesis, 2016.

[31] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *OSDI*, vol. 99, 1999, pp. 173–186.

[32] C. Cachin, "Architecture of the Hyperledger blockchain fabric," in *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, 2016.

[33] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *Annual International Cryptology Conference*. Springer, 1992, pp. 139–147.

[34] R. C. Merkle, "Protocols for Public Key Cryptosystems." IEEE, Apr. 1980, pp. 122–122. [Online]. Available: http://ieeexplore.ieee.org/document/6233691/

[35] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake," *self-published paper, August*, vol. 19, 2012.

[36] "Proof of Stake FAQ Âů ethereum/wiki Wiki." [Online]. Available: https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ

[37] V. Buterin, "Slasher: A Punitive Proof-of-Stake Algorithm," Jan. 2014. [Online]. Available: https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm/

[38] N. Szabo, "Smart Contracts: Building Blocks for Digital Markets." [Online]. Available: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html

[39] "Introduction to Smart Contracts âĂŤ Solidity 0.4.21 documentation." [Online]. Available: https://solidity.readthedocs.io/en/develop/introduction-to-smart-contracts.html#overview

[40] M. VukoliÄĞ, "Rethinking Permissioned Blockchains." ACM Press, 2017, pp. 3–7. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3055518.3055526

[41] K. Christidis and M. Devetsikiotis, "Blockchains and Smart Contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[42] K. R. ÃŰzyÄślmaz and A. Yurdakul, "Integrating low-power IoT devices to a blockchain-based infrastructure: work-in-progress." ACM Press, 2017, pp. 1–2. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3125503.3125628

[43] M. S. Ali, K. Dolui, and F. Antonelli, "IoT data privacy via blockchains and IPFS." ACM Press, 2017, pp. 1–7. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3131542.3131563

[44] H. Shafagh, L. Burkhalter, A. Hithnawi, and S. Duquennoy, "Towards Blockchain-based Auditable Storage and Sharing of IoT Data." ACM Press, 2017, pp. 45–50. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3140649.3140656

[45] S. Ibba, A. Pinna, M. Seu, and F. E. Pani, "CitySense: blockchain-oriented smart cities," in *Proceedings of the XP2017 Scientific Workshops*. ACM, 2017, p. 12.

[46] P. Missier, S. Bajoudah, A. Capossele, A. Gaglione, and M. Nati, "Mind my value: a decentralized infrastructure for fair and trusted IoT data trading." ACM Press, 2017, pp. 1–8. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3131542.3131564

[47] A. Dorri, S. S. Kanhere, and R. Jurdak, "Towards an Optimized BlockChain for IoT." ACM Press, 2017, pp. 173–178. [Online]. Available: http://dl.acm.org/citation.cfm?doid=

3054977.3055003