A Blockchain-Based Micro Economy Platform for Distributed Infrastructure Initiatives

Jan Kramer*†, Jan Martijn E. M. van der Werf*, Johan Stokking†, and Marcela Ruiz*
*Utrecht University, Princetonplein 5, 3584 CC Utrecht, the Netherlands
Email: {j.m.e.m.vanderwerf, m.ruiz}@uu.nl
†The Things Network, Rigakade 10, 1013 BC Amsterdam, the Netherlands
Email: jan@jankramer.eu, johan@thethingsnetwork.org

***Abstract*—Distributed Infrastructure Initiatives (DIIs) are communities that collaboratively produce and consume infrastructure. To develop a healthy ecosystem, DIIs require an economic model that balances supply and demand, but there is currently a lack of tooling to support implementing these. In this research, we propose an architecture for a platform that enables DIIs to implement such models, focused around a digital currency based on blockchain technology. The currency is issued according to the amount participants contribute to the initiative, which is quantified based on operational metrics gathered from the infrastructure. Furthermore, the platform enables participants to deploy smart contracts which encode self-enforcing agreements about the infrastructure services they exchange. The architecture has been evaluated through a case study at TTN a global distributed crowdsourced Internet of Things initiative. The case study revealed that the architecture is effective for the selected case at TTN. In addition, the results motivate future research lines to support scalability (i.e., to deploy the architecture on a larger scale) and security.**

Keywords*—Software architecture, blockchain, smart contract, digital currency, reward system.

I. Introduction

In recent years, our attitudes towards consumption and production have shifted towards a more distributed, peer-to-peer and sharing economic model [1]. Examples of drivers for this shift include globalization and the consumerization of digital technologies [2]. However, while platforms such as Airbnb, Kickstarter, and Etsy are indeed based around a peer-to-peer economy, they are still fully dependent on central organizations to manage their platforms. In the utopia of a true peer-to-peer economy, these dependencies would also be eliminated and replaced by a fully distributed alternative.

In addition to consumer goods and services, it is also possible to produce infrastructural services in a decentralized manner, as shown by The Things Network (TTN), a global, distributed, crowdsourced Internet of Things network initiative [3]. To refer to this type of initiative, we define the term Distributed Infrastructure Initiative (DII) as a group of individuals and organizations that cooperatively produce and consume a shared set of infrastructure services, without a centralized governance body.

A notable attribute of DIIs is that the participants themselves are tasked with producing the infrastructure, whereas traditionally, corporations such as telecom operators bear this responsibility. While corporations are incentivized by profits to produce the infrastructure, in DIIs there is no built-in incentive for participants to produce beyond their own need, especially in situations where the infrastructure is offered free of charge. Therefore, our main research question is: How to reinforce the participation in DIIs, and support a stable supply and demand in DDIs?

We propose a platform specific for DIIs to encourage participation by means of a rewarding system [4]. However, while there is a vast body of literature on related topics such as reward systems, We observe little practical tooling to enable DIIs develop such micro economies.

In this paper we present the design and evaluation of a software architecture that provides the tools to operationalise an economic model to encourage the particpants of DIIs. The economic model is based on a rewarding system to mitigate the uncertainties of current donation and need-based approaches. To achieve a fully decentralised architecture, we make use of the blockchain technology; which facilitates the specification of distributed ledgers, and a core logic by using smart contracts. For the evaluation, we conduct a case study in the context of a real DII at TTN.

The remainder of this paper is structured as follows. Section II provides some background on distributed ledger technology which is applied in the micro economy platform introduced in Section III. The case study is described in Section IV, and in Section V we discuss the findings of our research. Finally, Section VI concludes this paper and describes several areas for future research.

II. Background

A. Distributed Ledgers

Distributed ledgers are an emergent topic and have received a lot of attention recently due to the popularization of the concept of *blockchain* and its use in digital currencies such as Bitcoin. Essentially, a distributed ledger is a replicated, shared and distributed database which enables consensus between parties that do not trust each other [5]. The database consists of an append-only sequence of immutable transactions. As such, once a transaction has been confirmed, it is not possible to modify it. This makes it very suitable for applications which need a tamper-resistant data store, e.g. digital currencies or the micro economy platform we propose in this research.

A *blockchain* is a specific type of distributed ledger which groups transactions in blocks that are organized in a linked list, i.e. a chain of blocks. Blocks are brought into existence by

participants of the network through a process called "mining". Mining is intentionally made expensive so that it is economically infeasible for participants to forge data in order to gain an advantage. As such, mining can be used as a decentralized *consensus mechanism*. The actual implementation of consensus mechanisms differs between various types of ledgers, which is discussed in more detail in Section II-B.

Note that the term 'blockchain' is overloaded and can be referred to as either the generic architectural pattern that was popularized by its application in the digital peer-to-peer currency Bitcoin [6], or to the actual instantiation of the pattern as applied in projects such as Bitcoin and Ethereum [7]. For the purpose of this research, we are mainly interested in the generic architectural pattern of a distributed ledger.

B. Consensus Mechanisms

The consensus mechanism is core to a distributed ledger given that it provides its primary function: reaching consensus between parties about the "true" state of the ledger. The first ledger that was popularized, the Bitcoin blockchain, uses a *Proof-of-Work* (Pow) consensus mechanism [6]. PoW relies on participants continuously competing to solve puzzles, where the solution of each puzzle represents the missing piece of the next block. Since this process is computationally intensive, finding the next block and hence defining the upcoming state of the blockchain is expensive. Under the assumption that there is not a single party that operates more than half of the total mining resources, it is not possible for a single party to record false transactions, e.g. to *double spend* tokens.

The puzzle that is solved can have many forms. In the case of most digital currencies it consists of finding a hash that satisfies a specific property, i.e. the first n-bytes of the hash must be 0, where n represents the difficulty of the problem. For example, if the input data derived from the transactions is "0x1234" and n equals 2, then the puzzle is to find a value for i where the hash over "0x1234i" starts with "0x00...". While it is computationally intensive to find a correct hash, it is very easy to verify whether a hash is correct. This makes it very suitable as a means for other nodes to validate new blocks from other nodes.

One of the main drawbacks of the PoW approach is that the computations require a significant amount of energy resources. Some researchers estimate that a PoW network at scale would incur a 2.1% increase in carbon dioxide emissions worldwide [8]. Alternative approaches that do not incur a severe pressure on the environment include Proof-of-Space [9]–[11], which is based on miners providing disk space instead of computational resources, Proof-of-Stake [12], where miners have to put up a deposit – or *stake* – that can be burnt if they misbehave, and finally Proof-of-Authority, where a consortium of trusted parties is chosen upfront that are allowed to mine new blocks. While partially decentralized, this approach is not as open, given that not every participant can arbitrarily start mining.

C. Smart Contracts

Another aspect that varies across distributed ledgers is whether they support *smart contracts*. While popularized by Ethereum [13], Smart Contracts were actually first defined in [14]. A smart contract can be defined as the formalization of an agreement over a public network between parties that do not necessarily trust each other. It consists of promises that can be executed automatically, based on future inputs. The automatic execution allows anonymous parties to engage in transactions without a trusted third party being present. Examples of use cases include crowdfunding, content rights management, and escrow services.

In Ethereum, smart contracts are executed as part of transactions on the Ethereum Virtual Machine (EVM), a quasi-Turing-complete virtual state machine [7], [13]. The *quasi*-qualifier stems from the fact that transactions are limited by the amount of *gas* the sender provided to execute the transaction. Gas is a measure for the computational size of a transaction, and is consumed by every instruction the EVM executes (e.g. performing a calculation or writing/reading to or from permanent storage). Therefore, a sender has to provide sufficient gas for every step to execute. Since gas is provided by supplying additional Ether to the transaction, which has a real-world cost, this mechanism provides a safeguard against very large or inefficient transactions on the EVM.

D. Blockchain-Free Distributed Ledgers

Although the term 'blockchain' has been popularized, a more correct term for most use cases would be *distributed ledger*, since using a blockchain as data structure is merely an implementation detail of a system that tries to provide consensus in a trustless distributed setting. The fact that several other projects [15]–[17] have proposed an alternative data structure to store transactions supports this claim.

For example, IOTA uses a Directed Acyclic Graph (DAG) instead of a blockchain [17]. Each node in the DAG represents a transaction and each edge a reference to an earlier transaction. In order to publish new transactions on the network, a user has to perform PoW that includes data from the earlier transactions. By providing the PoW and publishing the transaction, the previous transactions are verified. Note that instead of depending on a separate group of miners, in IOTA, the users who engage in transactions verify transactions of other users. Therefore, IOTA also does not have transaction fees as in Bitcoin and Ethereum, although performing the PoW is computationally intensive and could be considered as implicit transaction costs.

One of the main benefits of this approach is scalability. In a blockchain-based ledger, every transaction has to be processed in order by every node since there is a single sequence of transactions. Due to its structure, a DAG-based ledger allows the network to temporarily diverge and therefore accept transactions asynchronously, which in turn leads to higher throughput.

However, at the time of writing, the distributed ledgers built using alternative data structures are still relatively immature and have to be validated by large scale real world usage.

III. Micro Economy Platform

The main purpose of the micro economy platform proposed in this research is to enable DIIs incentivize their participants to contribute to a shared infrastructure. It aims to provide these

incentives through a micro economy where participants can earn tokens in a DII-specific currency by contributing to the infrastructure. Additionally, participants can use the currency to exchange additional services with each other.

In principle, every *participant* of the DII is a potential stakeholder in the system. Participants can be both organizations as well as individuals, and among them we can distinguish two types. First, *contributors* are participants who add value by contributing to the infrastructure. Second, *users* are participants that utilize the infrastructure. These two types are not mutually exclusive, i.e. a participant can simultaneously be a contributor and a user. Finally, a potential third group of stakeholders are *investors*. Since the platform introduces an asset that represents some value and can be exchanged freely, it is possible that the asset attracts investors similar as to how investors hold Bitcoin and other digital currencies.

From a technical perspective, at the core of the DII is the infrastructure which consists of components that collect data about their operations, e.g. performance metrics or statistics about the amount of usage. This will form the basis for the integration with the proposed platform, as discussed in the next sections.

A. Requirements

The following functional (FR) and non-functional (NFR) requirements describe the features the platform should provide.

FR1 – Contributor ranking In order to reward contributions, we must know how *much* to reward and hence need to quantify a user's contributions. To that end, we introduce the concept of a Karma score, which is based on the metrics the platform collects from infrastructure components. The Karma score should be computed roughly along the following lines:

1) Infrastructure components continuously submit metrics to the platform;
2) Periodically (e.g. hourly), these metrics are aggregated per component, and converted to a single score using a function that is configurable per component type;
3) Based on the past n scores, per component a moving average is computed;
4) The overall Karma score of a participant is finally computed as the sum of the moving averages of all individual component scores.

FR2 – Issue tokens At each interval, after the Karma scores have been computed, the platform should issue a fixed number of new tokens in the DII-specific currency, *Wavelets* in the context of The Things Network. The tokens should be distributed to all contributors, proportional to their Karma scores.

1) After all Karma scores have been updated, compute the number of tokens to issue to every contributor by calculating their percentage of Karma and multiply that with the fixed total reward;
2) Issue the computed number of tokens to the contributors' wallets.

FR3 – Set up wallet To start receiving tokens, a participant needs to set up a wallet which is linked to a user account in the existing infrastructure:

1) The participant initializes a new wallet (client-side);
2) The participant sends a request to the platform to link the address of the wallet to the user account in the exiting infrastructure;
3) The platform requests the participant to follow an authorization flow to verify the participant's identity;
4) Only on successful authorization, the wallet address is linked to the user account.

FR4 – Exchange tokens

To allow participants freely exchange tokens of the DII-specific currency, the platform needs to support arbitrary transactions between wallets:

1) A participant with wallet address a issues a request to send n tokens to a given address b;
2) The platform checks whether the participant has enough tokens of the DII-specific currency;
3) If that is the case, the platform decreases the balance of the wallet with address a with n tokens, and increases the balance of the wallet with address b with n tokens.

FR5 – Marketplace

To enable participants set up self-enforcing agreements about network services and their usage, the platform should allow participants to deploy contracts that are executed periodically with aggregated network metrics, and can hold and transfer tokens. An example of such contract is further described in Section III-D2. To enable participants discover such offerings, the services should be listed in a marketplace:

1) A participant defines and deploys a smart contract on the platform which implements an interface that receives network metrics and executes arbitrary logic;
2) After the smart contract is deployed, the participant adds the service to the market place;
3) A participant interested in the service subscribes by making the required payment to the smart contract;
4) Periodically, when the smart contract is invoked, the predefined logic is executed;
5) The seller can discontinue the contract, which will trigger its removal from the marketplace and shut down the contract after a predefined period of notice;

NFR1 – Efficient Batch Handling The platform processes metrics in batches, and therefore the running time of a single batch should not exceed the interval between batches. An important factor to the running time is the number of metrics to be processed, which is dependent on the number of infrastructure components. While for the initial proof of concept, this number is relatively small, i.e. in the order of magnitude of several thousands, the platform should be able to scale towards hundreds of thousands of components.

NFR2 – Distributed Deployment One of the strengths of DIIs is that they do not have a central authority. However, this also means that they lack a single party they entrust with managing the state of the micro economy platform. Therefore, the platform needs to be operated in a distributed fashion by different parties.

NFR3 – Security Since DIIs are open for anyone to join, this does not exclude malicious actors. Especially due to the

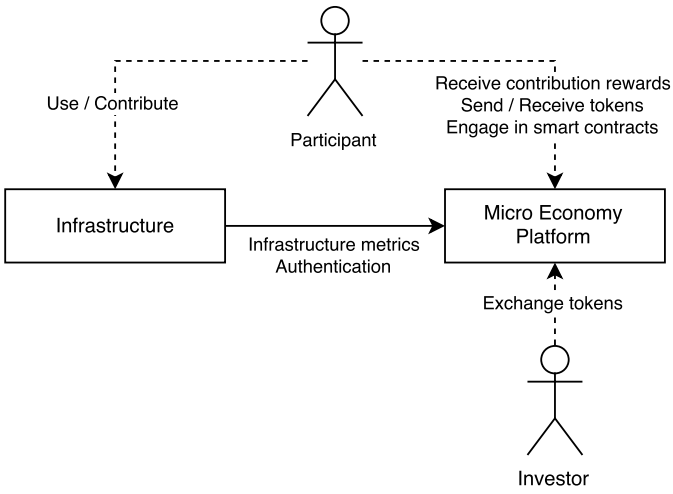

Fig. 1. Context diagram of micro economy platform

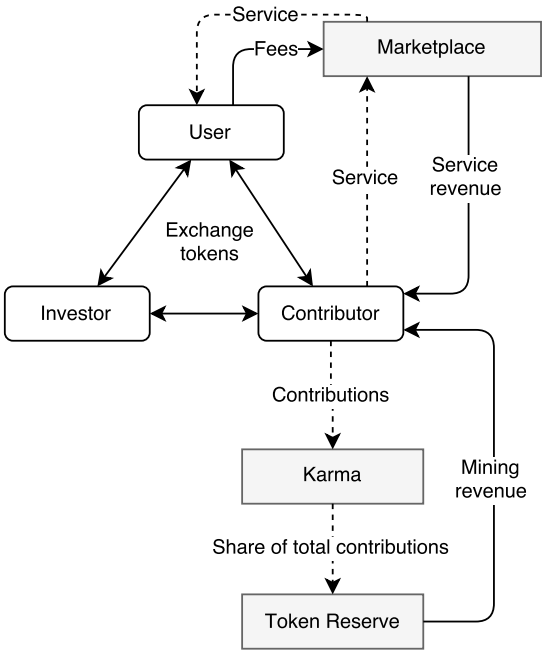

Fig. 2. Economic model as applied in the micro economy platform

fact that the platform can be used for economic gains, it is essential that there are safeguards in place that protect benevolent participants from attacks of participants with malignant intents.

B. Architecture

Following the requirements, we describe the architecture from several viewpoints based on the method described in [18].

1) Context: Figure 1 shows the micro economy platform as a black box in its context. In essence, it fulfills the following tasks. First, it accepts metrics about the infrastructure operations. Additionally, it exposes an authentication endpoint to link user accounts between the two systems. Second, based on the infrastructure metrics, contributors are rewarded with tokens, which can be exchanged with other participants. Third, the platform offers participants the ability to engage in smart contracts, e.g. an SLA as discussed in more detail in Section III-D2. Finally, a third group of outside actors, investors, might exchange tokens with participants without actively participating in the network.

The economic model employed by the micro economy platform is summarized in Figure 2. Fundamental to the model is the *token reserve*, a smart contract that holds and manages the tokens that are in circulation. It is also the only element that is able to issue new tokens, and therefore can be compared to a central bank.

The token reserve periodically issues a *mining revenue*. On a periodic basis, e.g., an hourly schedule, a fixed number of tokens is created and subsequently divided over all contributors, proportional to the size of their contributions. This process is analogous to mining in PoW-based cryptocurrencies, but instead of hashing power, this model allows any service or good to count as a contribution, as long as it can ultimately be quantified.

Another potential source of tokens for contributors is a marketplace where they can offer specific services to users. For example, a gateway owner could offer guarantees on a particular service level (e.g. 99.99% uptime) through an SLA, in exchange for a number of tokens per time unit. To prevent the gateway owner from violating the agreement, a possible penalty could be burning a pre-deposited amount of tokens. Given that the infrastructure already provides metrics to the platform, the contract could even be made self-enforcing by evaluating these metrics against predefined conditions. This example is discussed in more detail in Section III-D2. Note that this is just one example of a possible smart contract between network participants. Additionally, since smart contracts are essentially programs that can be submitted to the blockchain by any participant, anyone could define their own smart contracts in which they can record self-enforcing agreements with other participants.

Finally, as we have seen with other digital currencies, it is possible that the tokens attract investors who then start trading the token. By trading against other digital currencies or fiat, the token will gain value in the real world, which allows contributors and users to put an actual price on their services and contributions.

C. Functional Structure

The high-level platform design is depicted in Figure 3 by listing the main platform components and their relationships.

Infrastructure components are instrumented to submit performance and usage metrics to a *monitor*. The monitor subsequently stores the metrics temporarily. On a periodic basis, the *batch controller* triggers all monitors to submit their aggregated metrics to the *infrastructure metric store*. The platform aggregates the metrics to improve scalability, since storing every individual metric would cause a significant overhead.

Since the metrics have to be provided by an external source, it is important that we are able to trust the agent providing the metrics. In order to accomplish this, the DII only accepts data originating from known infrastructure components and monitors, which are registered in the *whitelist*.

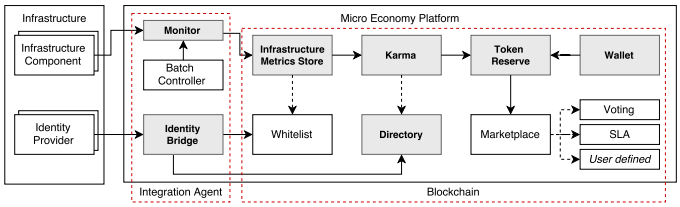

Fig. 3. High-level platform structure. Implemented components are highlighted (e.g. Monitor)

Based on the metrics, the *karma* component calculates a score for all participants that represent the significance of their contributions. Based on this score, the *token reserve* issues a number of tokens to the contributors *wallets* following requirement FR2.

The *directory* keeps track of the mapping between contributors and their wallet addresses, which is necessary to know where to issue new tokens. To register a wallet address, new participants have to perform a one-time action where they link their address to their infrastructure user account following requirement FR3.

Finally, participants can use their tokens to purchase services from other participants in a *marketplace*.

D. Information Flow

An important aspect in the market place are the token reward workflow and the way users can define SLAs.

1) Token Reward Workflow: Figure 4 shows in more detail how the token reward process as previously mentioned is executed by the various components.

Infrastructure components continuously report metrics about the usage and performance of the infrastructure to a monitor, which stores it in a temporary event store. On a predefined interval, e.g. hourly, one of the authoritative agents signals the monitors to start their batch process, which queries the event store and submits aggregated metrics to the infrastructure metrics contract on the blockchain. This contract subsequently performs various checks, before actually storing the metrics. Firstly it ensures that the source of the metrics is in fact allowed to submit metrics, and it verifies that the batch has not been sealed yet, which would mean the time window to submit the metrics had expired.

Note that some precision is lost by aggregating the metrics, but storing every metric separately would be unfeasible due to constraints in throughput and storage.

As soon as the finalization process is triggered, the batch is sealed by the metrics contract, and the registered batch listeners are triggered. An example of such listener is the Karma contract, which updates the karma scores of the users based on the newly added metrics, and calculates the token rewards that are to be distributed. Other examples could be user-defined contracts which also depend on metrics to execute their logic. An example of such contract is given in the next section.

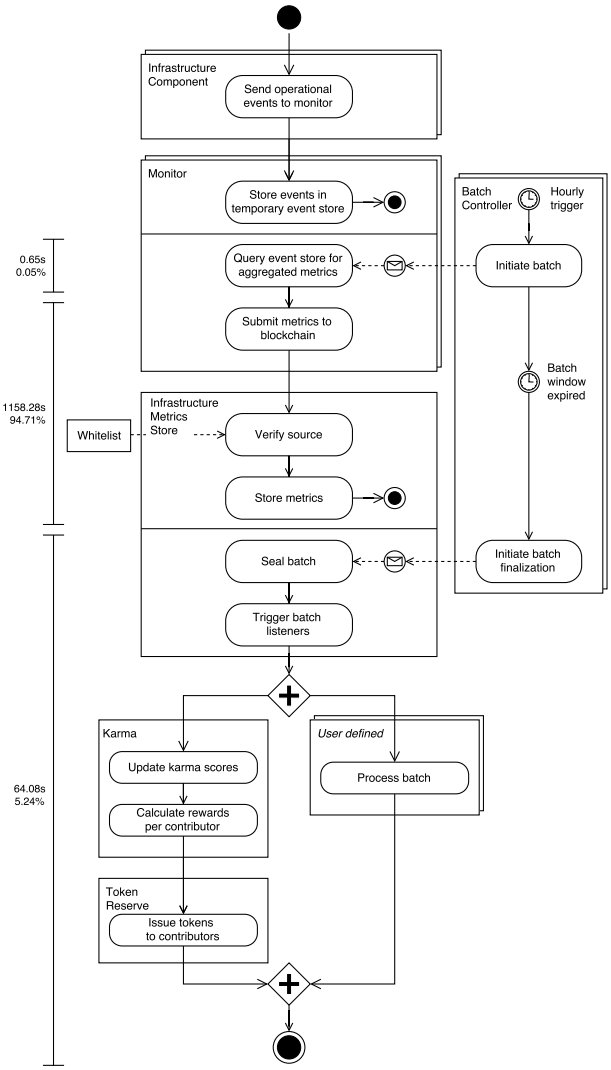

Fig. 4. Token reward workflow

2) User Defined SLA Workflow: Figure 5 describes how a user defined SLA could work in practice. It would ultimately be up to users themselves to define these contracts, although the platform could provide template contracts.

In this example, the contract starts with a contributor deploying a SLA smart contract. The contract contains some parameters, e.g. an uptime percentage and a price per month. The contract subsequently registers itself as a listener for new metrics with the infrastructure metrics contract, and the contributor adds the service to a marketplace through which users can subscribe. A subscription is started when a user deposits its first payment, which is kept in escrow by the SLA contract.

Then, on every batch, the SLA contract receives a signal and queries the infrastructure metrics component for the relevant metrics. If the metrics meet the pre-configured criteria then the contract pays out the tokens from all active subscribers for the current cycle to the contributor, and disables the subscriptions which do not have sufficient funds left to pay for another cycle. In case the metrics do not meet the pre-configured criteria, the remaining tokens are returned to

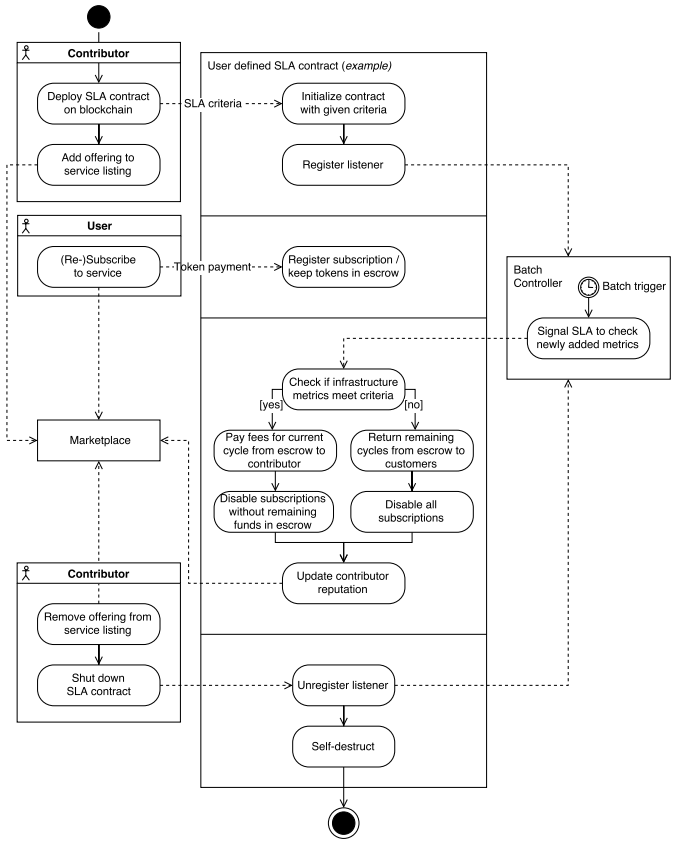

Fig. 5. User defined SLA workflow

the users and the subscriptions are cancelled. Finally, the reputation of the contributor is updated either positively or negatively depending on the SLA outcome. The reputation is shown on the marketplace and can provide users with insight in the historical performance of the selling contributors.

Ultimately, a contributor can choose to sunset its service which entails removing it from the marketplace and shutting down the contract. This will unregister the listener and self-destruct the service. Any remaining tokens in escrow are returned to the users.

E. Runtime Environment

Figure 6 depicts the runtime environment of the various platform components. Central to this view is the blockchain, which is represented as a logical component deployed on multiple physical nodes. Most of the platform modules are in fact deployed as smart contracts on this blockchain, since they operate on state that has to be shared across many nodes that lack a fully trusting relationship. All interaction with these smart contracts is performed through *blockchain clients*, which is the third party software that runs the blockchain. Note that the term *client* does not refer to the client–server architectural pattern. Instead, the blockchain is a peer-to-peer network and the client is used to operate a node in this network.

While all nodes participate in the blockchain network, only the *authoritative nodes* are allowed to validate transactions and issue new blocks. In addition to validating new blocks, they host the identity bridge module, which provides integration with the identity provider (i.e. user directory) of the existing infrastructure. The identity bridge must be deployed on trusted nodes, because the platform needs to securely verify the identity of participants, and the authoritative nodes are the only nodes that can be fully trusted. The authorities are chosen during the initial set-up of the blockchain by putting their addresses in the blockchain configuration. This configuration is subsequently shared among all authorities and must be used to successfully join the blockchain network. At a later stage, if a new authority wants to join or an existing authority must leave, a majority of the authorities must update their configuration, after which the new list becomes active.

Each monitor is deployed on a contributor node. Since the number of infrastructure components can grow beyond the number a single monitor can handle, the monitors should scale horizontally. Each monitor instance also has an event store which is a simple database that is used as a buffer to temporarily store metrics until they have been submitted to the infrastructure metrics store.

Finally, the wallet software is client-side which is necessary to keep the platform distributed and hence runs on every user's own device. Similar to the *identity bridge* and *monitor*, it communicates with the rest of the blockchain network through a *blockchain client*.

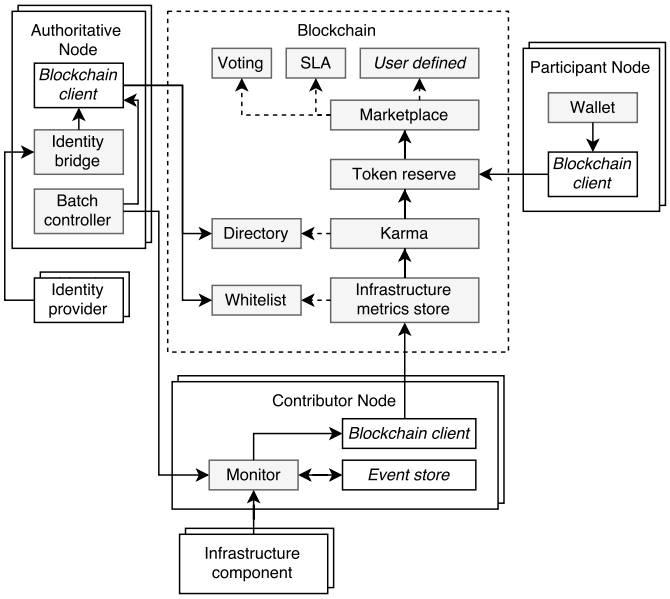

Fig. 6. Runtime environment

IV. Case Study

To evaluate the proposed architecture, we conducted a case study at TTN. Figure 7 depicts the typical usage of the IoT network infrastructure provided by TTN. The sequence is triggered by an IoT device (e.g. a sensor) transmitting an uplink message (1) which is received by zero or more gateways. Each gateway forwards the message to the router it is connected to (2), which in turn routes the message to a broker (3). The broker subsequently deduplicates the set of received messages belonging together, does a lookup to determine the application the message belongs to, and forwards the message to the corresponding handler (4). The handler then decrypts

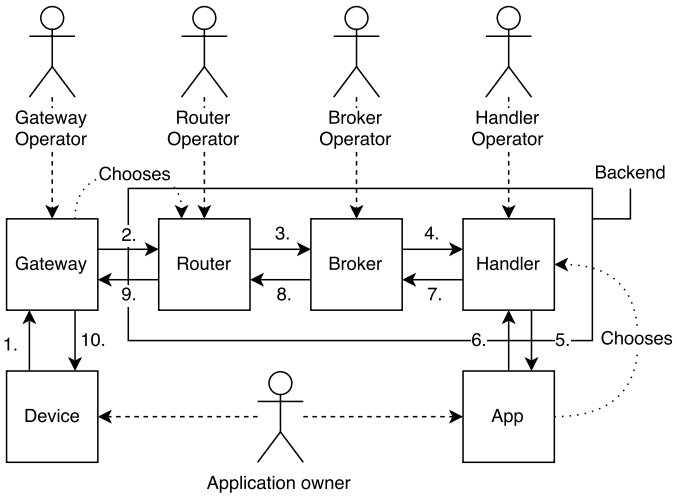

Fig. 7. Typical usage scenario of The Things Network

and decodes the payload and publishes the message to the application (5).

In case the application has scheduled a downlink (6) message, the handler encodes and encrypts the corresponding payload and sends it to the broker (7) which forwards it to the router that is connected to the gateway that has been selected as the best downlink option based on signal strength and utilization (8). Finally, the router schedules the downlink for the selected gateway (9) which transmits the message to the device (10).

As the usage scenario shows, there are many roles involved in using and operating the infrastructure. The gateway operator has to purchase a gateway, install it at a proper location (e.g. high altitude, outside, etc.) and provide it with electricity and internet connectivity. The routing service providers (router/broker/handler operators) have to operate a server that runs the TTN backend components and make sure everything is kept healthy and up to date. In short, these roles "deposit" value by contributing infrastructure, whereas, application owners only *use* the network and thereby "withdraw" value.

A. Proof-of-Concept Implementation

In order to validate the design of the architecture, we developed a proof-of-concept (PoC) of the micro economy platform. The PoC does not implement the full architecture, but is restricted to a subset of components that we deemed necessary to validate the essential aspects of the concept.

The gray components in Figure 3 have been implemented, the others have been omitted in the PoC. In essence, the PoC consists of two high-level components. The first component being a *blockchain* implementation with on top a set of smart contracts, and secondly an *integration agent* that links the blockchain and existing infrastructure.

1) Integration Agent: The monitor and identity bridge components have been developed in one software component, the *integration agent*, for ease of development and deployment. The main responsibility of the monitor is ensuring operational metrics are collected and submitted to the rest of the micro economy platform. Secondly, the identity bridge provides the integration of the existing user base with the blockchain user infrastructure.

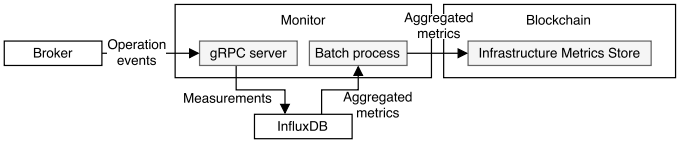

Fig. 8. Monitor implementation

Monitor Figure 8 depicts a more detailed view of the implemented monitor. Given that in the context of TTN infrastructure components already expose data over gRPC[1] streams, and bindings for these streams are available in the Go programming language[2], we chose to implement a gRPC server in Go. The infrastructure components are configured to send events about their operation to the monitor, which are then temporarily stored as measurements in InfluxDB[3], a time-series database. A time-series database, and specifically InfluxDB, is appropriate here, since it allows for easy aggregation over time and has built-in support for retention policies to automatically discard old data.

The batch process that submits the aggregated metrics to the blockchain runs in a separate thread parallel to the gRPC server. This process is triggered on an hourly basis, and aggregates the number of messages and total airtime, i.e. the duration of the gateway being active to send or receive a message, over the previous period through a query on the InfluxDB data store.

Note that even though we have only deployed a single monitor in the current proof-of-concept, the architecture prescribes that ultimately many monitors are deployed and they all collect and report metrics. This is necessary to distribute the load when more infrastructure components join the network, but also introduces the need for an additional component that coordinates the various monitors. This task is delegated to the *batch controller*, which has been omitted in the PoC.

Identity Bridge Both the existing infrastructure and blockchain have their own security system. Since we need to identify users according to their TTN credentials, for example to know where to send rewards, we need to provide an integration between the two systems.

The TTN security system is a bespoke software component, but is based on standard protocols and offers integration facilities through OAuth. The blockchain has, due to its distributed nature, a slightly different approach to security based on public key cryptography. Users need to generate a wallet, which in essence is a private key. To submit transactions from that wallet, a user needs to sign the transaction using the private key, and only with a proper signature will the transaction be accepted by other blockchain nodes.

As depicted in Figure 9, to integrate the two systems, the identity bridge component requests users to follow the OAuth flow of the TTN security system, and subsequently provide their wallet address. The identity bridge then registers

[1]See https://grpc.io/about/

[2]See https://golang.org/

[3]See https://github.com/influxdata/influxdb

the username-address combination on the blockchain in the Directory contract, after which the identification procedure is finished.

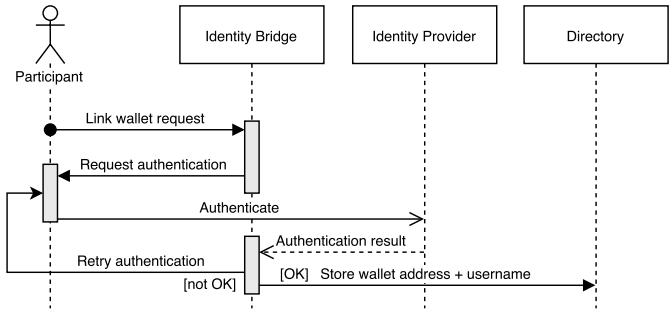

Fig. 9. Sequence diagram of wallet registration

2) Blockchain: Most of the core functionality has been implemented through smart contracts on the private blockchain. There are two concepts of importance here: the underlying blockchain itself, i.e. the infrastructure, and the smart contracts we implemented on top.

Ethereum implementation Although there are many different blockchain technologies available, for this PoC we opted for a private Ethereum instance. Our main reason to choose Ethereum is its ability to deploy and execute smart contracts that contain arbitrary logic through the Ethereum Virtual Machine (EVM). Smart contracts for Ethereum are written in Solidity, a strongly typed language of which the syntax closely resembles those of general purpose languages such as Java and C#. Solidity is compiled to EVM-specific bytecode, so in theory, other languages could be developed to build smart contracts for EVM-enabled blockchains. An example of a smart contract implementing the tokens exchanged between participants is shown in Figure IV-A2.

A smart contract in Solidity is similar to the concept of a class, i.e. it has a constructor, methods and properties. Deploying a smart contract subsequently resembles instantiating a class, and consists of compiling the contract to EVM bytecode and attaching it to a transaction. When the contract has been deployed, i.e. the next block is mined, the smart contract is assigned a unique address. Read operations on a smart contract can occur without any transactions. One has to simply inspect the state of the blockchain at the address where the smart contract is deployed. However, when performing operations that manipulate the state of a smart contract, one needs to send a transaction to the smart contract with the operation encoded in bytecode.

Instead of using the public Ethereum chain, the PoC uses a private instance. A deployment on the public blockchain would not be cost-effective, since storing a relative high amount of data is expensive, i.e. in the order of magnitude of hundreds or thousands of USD per hour. Additionally, by having a private blockchain, we have more control over the configuration and are thereby able to tune parameters such as block time to maximize the performance for our specific case, which would not be possible in a public blockchain.

Figure 11 depicts the actually implemented smart contracts as a simplified UML class diagram. The full implementation can be found in [19].

```
contract Token {
    address owner;
    mapping(address => uint) balances;

    function Token() {
        // `msg.sender`: user deploying the contract
        owner = msg.sender;
    }
    // Issue new token
    function create(address recipient, uint amount) {
        // Only allowed by the owner of the token contract
        if (msg.sender != owner) { throw; }
        balances[recipient] += amount;
    }
    // Transfer tokens to another user
    function transfer(address to, uint amount) {
        if (balances[msg.sender] < amount) { throw; }
        balances[msg.sender] -= amount;
        balances[to] += amount;
    }
}
```

Fig. 10. Example of smart contract in Solidity

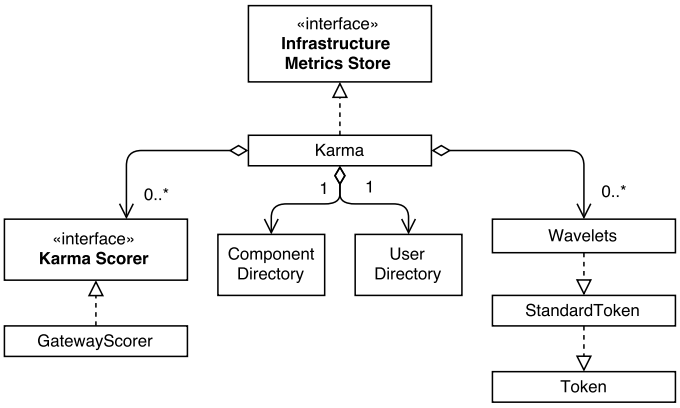

Fig. 11. Overview of smart contracts and their relationships

Although the architecture suggest a separation between the Karma and Infrastructure Metrics Store contracts (cf Figure 3), the functionality of both contracts has been implemented in the Karma contract for the purpose of this PoC. The main reason was to reduce cross-contract communication, which made it both easier to implement as well as resulted in better performance. The reward flow, as previously described in Section III-D1, therefore concretely looks as follows.

First, from the monitor, new metrics are submitted to the Karma contract. Since every component type is different and hence has different sets of metrics, the Karma contract accepts an integer array of arbitrary length as input. For example, for a gateway, the metrics consist of the number of messages and total airtime, e.g. [2, 100, 3, 180] for "2 uplink messages, 100ms uplink airtime, 3 downlink messages, 180ms downlink airtime". Although currently not implemented, the metrics for a router would most likely not contain the airtime, but instead something that would be more representative of its performance such as average latency and uptime.

Secondly, we need to compute a single Karma score for every component, regardless of the types of metrics we receive. For this, we define a Karma Scorer interface, which takes as input an arbitrary set of metrics and outputs a single Karma score. For the PoC, only a GatewayScorer has been implemented. Due to the common interface it should be trivial to add support for other component types. The GatewayScorer implementation defines a number of tiers based on the amount of airtime a gateway has processed, where more airtime indicates a larger contribution and hence a higher reward.

After the Karma score has been calculated, the overall current Karma score of both the component and subsequently the user have to be updated given that they are moving averages of previous n periods. Finally, after the metrics have been updated and the batch finalization is initiated, the rewards are 'mined' and issued to all contributing participants. This entails calculating the total reward, and subsequently dividing a fixed number of Wavelets proportionally over the contributors.

B. Analysis

By implementing and deploying the PoC, we were able to analyze the architecture from a performance and security perspective.

1) Performance: The implementation of the PoC shows that the performance of the architecture is limited by the current use of a blockchain. Notably, a large share of the active running time is spent on submitting metrics to the blockchain.

We measured the time the PoC required to process a single batch by logging timestamps at various stages in the batch: 1) at the start, 2) after the metrics are aggregated, 3) after the metrics are submitted, and 4) after the batch is finalized. Figure 4 has been annotated to highlight the exact steps the stages encompass. The measurements were conducted during a 24-hour period, so in total 24 runs were measured. The number of infrastructure components active during this period ranged from 2,010 to 2,082. On average, a single run took 20 minutes and 23 seconds (SD = 5:05), of which 94.71% of the time was spent submitting metrics to the blockchain, 5.24% finalizing the batches, and only 0.05% aggregating the metrics.

Since the interval between batches is one hour, there is not much headroom to scale up in terms of number of tracked infrastructure components.

2) Storage: In addition to the computational time, another constraint is storage. After circa two months of running, the total storage required for a single node amounts to roughly 40GiB, and it will increase only more over time. Although the smart contracts only store metrics for a given window, currently Ethereum retains all data ever submitted to the blockchain. There are theoretical solutions to this problem, but none of them have been implemented. For example, in [20] a concept called State Tree Pruning is proposed where nodes from the state tree that are no longer in use can be removed. This would allow to keep a constant storage requirement for a constant number of infrastructure components. Another potential solution is proposed in [21], where only a few nodes need to retain the entire blockchain, and most nodes only need a significantly smaller blockchain, without reducing the security of the overall system. Unfortunately, none of these solutions have been implemented yet.

Another, more radical, solution would be to stop storing metrics on the blockchain altogether, and instead move to offchain storage, e.g. based on IPFS as described in [22]. This would mean that the actual data would be stored in a distributed filesystem, and only a reference would need to be stored on the blockchain. The obvious benefit would be that a solution such as IPFS is a much more efficient data store, but it would make integrating the metrics in scenarios such as the proposed user defined SLA smart contracts more complex.

3) Security: Given that rewards contributors receive value, it is necessary to make sure the system does not contain any loopholes that can be used by malicious actors to gain an unfair advantage. For most of the platform we use open-source software and standards which are well-maintained. Vulnerabilities might occur in these components, but can largely be mitigated by keeping the software up-to-date.

More pressing is that, in theory, a user could generate fake data and present it to the platform as being legit. For example, a user could implement a software gateway that generates messages. The platform would process these messages and assume that the particular user is contributing significantly and issue rewards accordingly. In the current set-up, we prevent this through the application of a *whitelist*. Every component first has to be whitelisted by other (trusted) users, before data from that component is accepted. However, there are two main drawbacks to this approach. Firstly, the solution does not provide full guarantees. As soon as a component is trusted, it can start generating fake data. The second drawback is that it requires a manual step before contributors can get rewarded, namely getting authorized by the whitelist. Ideally, the platform would provide a built-in mechanism to overcome this issue in an automated way, but this remains an open issue.

V. Discussion

A. Findings

1) Scalability: During the implementation of these smart contracts we quickly ran into the current technical limits of blockchain, especially in the area of scalability. The PoC developed during the case study showed us that both computational and storage requirements quickly become too high to be practical when scaling up. Note that this is not only a pressing issue for private distributed ledger implementations such as the one explored in our research, but also for well-known public blockchains such as Bitcoin and Ethereum. Noteworthy in that regard is the scaling debate Bitcoin is currently facing. Various groups within the ecosystem have different visions on how the underlying technology should be scaled, but up until the time of writing no consensus (ironically) has been achieved on which direction the community should pursue.

2) Off-Chain Assets: Another pain point for distributed ledgers relates to off-chain assets, i.e. data about "stuff" from the real world, as opposed to assets that live *on* the blockchain such as bitcoins. For on-chain assets, their validity and ownership is governed by built-in mechanisms, i.e. they only exist because the blockchain tells us so. However, for off-chain assets, someone first has to submit facts about the asset to the blockchain. Although that specific fact is securely stored on the blockchain from that point on, it does not prove anything about its truthfulness in the real world. Ultimately, everyone has to trust the original party to have provided valid data. Measures can be taken to improve the trustworthiness, e.g. by requiring a quorum to agree on the data before accepting it, but it still does not provide a watertight proof on truthfulness.

3) Degree of Trust: One should note that not every ecosystem is fully trustless. Currently, most of the major blockchains assume that all participants are anonymous and not necessarily to be trusted. However, this assumption does not hold for every community. For example, in the context of TTN we saw

that it was possible to identify a consortium of organizations that are widely recognized as being trustworthy. In addition to circumventing the previously discussed issue on off-chain data, this "semi-trustlessness" can be leveraged to use a less strict consensus mechanism such as Proof of Authority. This results in a lower operational cost, since it is no longer necessary to perform the mining as is the case for PoW. Naturally, some communities do require the system to be able to operate under the "trustlessness assumption", but it is nevertheless an important aspect to consider when designing a new system that employs a distributed ledger.

B. Threats To Validity

While we expect the platform and underlying concepts to be applicable to other DIIs, we only studied one case study which is a threat to the *external validity* of this research. Second, the implemented PoC is a subset of the proposed architecture, and some concessions have been made due to time constraints. Therefore, these discrepancies might threaten the *construct validity* of our research. Finally, while we have compared various distributed ledger technologies, we implemented the platform only on top of Ethereum. This may have led to a bias in our findings. Given more time, we would have explored different technologies.

VI. Conclusions and Future Work

The results of our research provide an architecture for a Micro Economy platform for DIIs. The outcomes of the case study conducted at TTN reveals an effective application of the architecture to a DIIs case. From the point of view of the TTN's stakeholders, the architecture is a stepping stone to support DIIs developments. In addition, the results of this research motivate an extensive study on how to incorporate security and scalability support.

First, more research is necessary to find mechanisms to *securely* collect metrics about infrastructure components, because the current architecture is not fully sealed against attacks where participants fake component data to gain an advantage. One solution we envision would be to apply cryptography to securely sign messages so their origin is warranted to be legitimate. Another possible solution is to use a system of witnesses that vote on the legitimacy of submitted data. Nevertheless, more research is required to validate both ideas and find possible alternative solutions.

Secondly, the implementation shows that the scalability of the current architecture is relatively limited due to the processing speed and storage requirements of the blockchain. While there is still room to optimize the current proof of concept, e.g. by tuning parameters such as block size and the interval between batches, we don't expect order of magnitude improvements. Therefore, it is necessary to fundamentally improve the performance of the architecture. Two ideas that need to be further investigated are 1) storing metrics off-chain (e.g. using IPFS [22]) to reduce the storage requirements, and 2) explore the possibility of using *payment channels* [23] for smart contracts to reduce the number of required transactions and thereby increasing the overall throughput.

In the short term, we plan to conduct a multi-case study to assess the current scalability support of the architecture. Based on the findings, we plan to release a new version that can be evaluated by the DIIs participants. Therefore, we find appropriate to conduct qualitative research as well. In this way, we want to advance towards an architectural pattern [24], [25] for distributed-ledger systems. On a final note, the landscape of distributed ledgers advances at a rapid pace and is of relatively tender age. It is therefore essential to keep track of developments in this research area and continuously assess the potential of new ideas and technologies.

References

[1] J. Hamari, M. Sjöklint, and A. Ukkonen, "The sharing economy: Why people participate in collaborative consumption," *JASIST*, vol. 67, no. 9, pp. 2047–2059, 2016.

[2] A. Sundararajan, "The power of connection: Peer-to-peer businesses," 2014.

[3] The things network. [Online]. Available: https://thethingsnetwork.org

[4] S. Kerr, "On the folly of rewarding a, while hoping for b," *Academy of Management Journal*, vol. 18, pp. 769–783, 1975.

[5] M. Swan, *Blockchain: Blueprint for a new economy*. "O'Reilly Media, Inc.", 2015.

[6] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[7] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," 2014.

[8] J. Becker, D. Breuker, T. Heide, J. Holler, H. P. Rauer, and R. Böhme, *Can We Afford Integrity by Proof-of-Work? Scenarios Inspired by the Bitcoin Currency*. Springer, 2013, pp. 135–156.

[9] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak, *Proofs of Space*. Springer, 2015, pp. 585–605.

[10] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz, "Permacoin: Repurposing bitcoin work for data preservation," in *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, May 2014.

[11] S. Park, K. Pietrzak, J. Alwen, G. Fuchsbauer, and P. Gazi, "Spacecoin: A cryptocurrency based on proofs of space," IACR Cryptology ePrint Archive, 2015: 528, Tech. Rep., 2015.

[12] I. Bentov, A. Gabizon, and A. Mizrahi, *Cryptocurrencies Without Proof of Work*. Springer, 2016, pp. 142–157.

[13] V. Buterin, "A next-generation smart contract and decentralized application platform," 2014.

[14] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, 1997.

[15] S. D. Lerner, "Dagcoin: a cryptocurrency without blocks," 2015.

[16] A. Churyumov, "Byteball: a decentralized system for transfer of value," 2015.

[17] S. Popov, "The tangle," 2016. [Online]. Available: https://iotatoken.com/IOTA_Whitepaper.pdf

[18] N. Rozanski and E. Woods, *Software systems architecture: working with stakeholders using viewpoints and perspectives*. Addison-Wesley, 2012.

[19] J. Kramer, "A blockchain-based micro economy platform for distributed infrastructure initiatives," 2017.

[20] "State tree pruning." [Online]. Available: https://blog.ethereum.org/2015/06/26/state-tree-pruning/

[21] D. Frey, M. X. Makkes, P.-L. Roman, F. Taïani, and S. Voulgaris, "Bringing secure bitcoin transactions to your smartphone," in *Adaptive and Reflective Middleware*. ACM, 2016, pp. 3:1–3:6.

[22] J. Benet, "IPFS - content addressed, versioned, P2P file system," *CoRR*, vol. abs/1407.3561, 2014. [Online]. Available: http://arxiv.org/abs/1407.3561

[23] C. Decker and R. Wattenhofer, *A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels*. Cham: Springer International Publishing, 2015, pp. 3–18.

[24] P. Avgeriou and U. Zdun, "Architectural patterns revisited – a pattern language," in *EuroPLoP 2005*.

[25] J. Peters, J. M. E. M. van der Werf, and J. Hage, "Architectural pattern definition for semantically rich modular architectures," in *WICSA 2016*. IEEE Computer Society, 2016, pp. 256–261.