MASTER THESIS

A BLOCKCHAIN-BASED MICRO ECONOMY PLATFORM FOR DISTRIBUTED INFRASTRUCTURE INITIATIVES

Jan Kramer Department of Information and Computing Sciences Utrecht University

Date: August 2017 Supervisors: dr.ir. J.M.E.M. van der Werf (UU) dr. L.M. Ruiz Carmona (UU) J. Stokking MSc (The Things Network) W. Giezeman MSc (The Thitngs Network)

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 such economic models. In this research, we propose an architecture for a platform that enables DIIs to implement such micro economic 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 validated through a case study at TTN, where a proof of concept of the architecture was implemented and evaluated. The case study revealed that the architecture is effective for the given situation, but needs more research in the areas of scalability and security to be deployed on a larger scale.

Acknowledgement

This thesis is the culmination of my graduation project for the Master Business Informatics program at Utrecht University. It would not have the shape it has today without the help of several key people. In particular, I would like to thank my supervisors who despite busy schedules were always able to provide me with invaluable insights. Jan Martijn, thank you for the pragmatism and out-of-thebox thinking you provided during our biweekly meetings. Johan, thank you for the freedom you offered me to conduct my research as I saw fit and the patience you showed along the way. Wienke, thank you for your endless enthusiasm, wise conversations and interesting book suggestions.

Finally, I would like to thank everyone else at The Things Network for the inspiring environment, and anyone I forgot to mention who provided input on my research over the past year.

– Jan Kramer

Contents

1	Intr	roduction 4	1							
	1.1	Case Subject	5							
	1.2	Running Example	7							
	1.3	Outline	7							
2	Res	earch Approach	3							
	2.1	Research Questions	3							
	2.2	Research Methods)							
		2.2.1 Literature Review)							
		2.2.2 Design Science)							
		2.2.3 Case Study)							
	2.3	Contributions)							
		2.3.1 Scientific)							
		2.3.2 Societal)							
3	The	Theoretical Background 11								
	3.1	Socio-Economic Perspective	1							
		3.1.1 Economic Principles	1							
		3.1.2 Traditional Buyer-Seller Market	2							
		3.1.3 Mining Rewards	2							
		3.1.4 Distributed Peer-to-Peer Marketplace	2							
	3.2	2 Software Architecture								
		3.2.1 Definitions	3							
		3.2.2 Viewpoints and Perspectives	3							
	3.3	Distributed Ledgers	5							
		3.3.1 Public vs Private vs Hybrid	5							
		3.3.2 Smart Contracts	5							
		3.3.3 Consensus Mechanisms	3							
		3.3.4 Blockchain-Free Distributed Ledgers	7							
4	Req	uirements for a Micro Economy Platform 19	•							
	4.1	Purpose & Scope)							
	4.2	Stakeholders)							
	Functional Requirements)								
	-	4.3.1 Reward Model)							
		4.3.2 Issue Tokens	1							
		4.3.3 Set Up Wallet	1							
		4.3.4 Send/Receive Tokens	2							
		1								

		4.3.5 Marketplace	. 22					
4.4 Non-Functional Requirements								
		4.4.1 Efficient Batch Handling	. 23					
		4.4.2 Distributed Deployment	. 23					
		4.4.3 Security \ldots	. 23					
		v						
5	Mic	o Economy Platform Architecture	24					
	5.1 Context Viewpoint							
		5.1.1 Context Diagram	. 24					
		5.1.2 Economic Model	. 26					
	5.2	Functional Viewpoint	. 27					
		5.2.1 High-Level Structure	. 27					
		5.2.2 Token Reward Workflow	. 27					
		5.2.3 User Defined SLA Workflow	. 28					
	5.3	Deployment Viewpoint	. 31					
	0.0	5.3.1 Buntime Environment	. 31					
		5.3.2 Technology Dependencies	31					
			. 01					
6	Eva	uation: Case Study at The Things Network	33					
	6.1	Proof-of-Concept Implementation	. 33					
		6.1.1 Integration Agent	. 34					
		6.1.2 Blockchain	. 35					
	6.2	Analysis	. 38					
	0	6.2.1 Scalability	. 38					
		6.2.2 Security	. 39					
		0.2.2 Sociality						
7	Dis	ussion	40					
	7.1	Findings	. 40					
		7.1.1 Scalability \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots	. 40					
		7.1.2 Off-Chain Assets	. 41					
		7.1.3 Degree of Trust	41					
	7.2	Threats To Validity	· 11 41					
	1.2		• •					
8	Cor	elusions	43					
	8.1	Results	. 43					
	8.2	Future Research	. 44					
	0.2							
Re	efere	ces	45					
\mathbf{A}	\mathbf{Sm}	rt Contracts	48					
В	Per	ormance measurements	54					
~	ъ		.					
С	Faper 55							

Chapter 1

Introduction

In recent years, our attitudes towards consumption and production have shifted towards a more distributed, peer-to-peer and sharing economic model [12]. Examples of drivers for this shift include globalization and the consumerization of digital technologies [25]. 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 [28]. 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, given basic economic principles, voluntary contributions to such initiatives are limited and potentially unsustainable.

Developing a micro economy specific to DIIs can address this issue by enabling incentives to be set up so that contributions are rewarded. However, while there is a vast body of literature on related topics such as reward systems, there is little practical tooling to enable DIIs develop such micro economies.

Therefore, in this research we aim to provide a software architecture of a platform that provides the tools necessary to operationalize an economic model that incentivizes participants to contribute beyond their own needs and altruism. To achieve a fully decentralized architecture, the platform is centered around a blockchain using smart contracts to facilitate the core logic. Finally, the architecture is validated by implementing a proof of concept and evaluating it in the context of TTN, a real-world DII.

1.1 Case Subject

To further illustrate the problem we aim to address, we introduce The Things Network (TTN) as our case subject. TTN aims to connect individuals and organizations with a common interest in IoT through an online platform, and enable them to cooperatively build the shared network.

Figure 1.1 pictures a brief overview of the main roles and components involved in the network. On an infrastructural level, the network consists of *gateways* which, analogous to access points in WiFi networks, provide edge connectivity to *IoT devices* in the field using radio technology. Each gateway connects to the *backend*, which ensures uplink and downlink messages between the IoT devices and applications are properly routed.

The initiative is distributed and each of the infrastructure components is deployed multiple times and operated by separate entities. Among these entities, we distinguish the following roles. *Application owners* create *IoT applications* that communicate with devices in the field, and are effectively the consumers of the infrastructure. *Gateway operators* subsequently provide network coverage by placing gateways, and *routing service providers* operate the backend components that ensure messages are routed properly. Note that these roles are not mutually exclusive. For example, a gateway owner can simultaneously own applications and host backend components.



Figure 1.1: Roles and components in The Things Network



Figure 1.2: Typical usage scenario of The Things Network

Figure 1.2 depicts the typical usage of the IoT network infrastructure. 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 message 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 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 *use* the network and thereby "withdraw" value.

1.2 Running Example

To further concretize the example, let us introduce a running example within the context of TTN.

Consider a fictional urban area where Parco, a parking garage company, wants to roll out an IoT application that keeps track of its parking space occupancy. However, while the software is developed in-house, they do not have the expertise to build and operate the necessary IoT network hardware as depicted in Figure 1.2.

Conveniently, a group of independent parties have already invested in the required network infrastructure. The group consists of TTN, the initiatives' foundation, Alice, a private individual, and Netcorp, a network infrastructure company. TTN operates a number of routing services (router, broker and handler), whereas Alice and Netcorp have invested in various gateways.

Finally, we introduce Bob, an individual who believes in a future where digital currencies such as Bitcoin and Ethereum play an important role and therefore invests in various digital currencies in his spare time.

In later chapters, where appropriate, concepts are further clarified using this running example in a textbox such as the following.

Example 1.1 If we apply the running example to Figure 1.2, then Parco is an example of an *application owner*, and their *app* the parking space tracker. The *devices* are represented by sensors deployed in the parking garage, and communicate through *gateways* operated by both Alice and Netcorp. The routing services (*router, broker* and *handler*), finally, are are operated by TTN and used to get the messages from the gateway to the parking space tracker and vice versa.

1.3 Outline

The remainder of this thesis is structured as follows. Chapter 2 describes the approach of this research, including its research questions and methods. Chapter 3 provides a theoretical background from the perspectives of socioeconomics, software architecture and distributed ledgers. In Chapter 4, we describe the functional and non-functional requirements of the micro economy platform proposed in this research, of which an architecture is provided in Chapter 5. The architecture is evaluated in Chapter 6, and the findings of designing and implementing the proof of concept are discussed in Chapter 7. Finally, we conclude the research in Chapter 8 by addressing the original research questions and describing areas for future research.

Chapter 2

Research Approach

2.1 Research Questions

The main research question this research aims to address reads as follows.

What is an *effective architecture* for a platform that enables *distributed infrastructure initiatives* to develop their own *micro economy*?

The following sub-questions further specify the research and support answering the main research question.

SQ1 What are appropriate methods to incentivize value adding activities?

Rationale: There is a vast body of knowledge on reward systems and incentives, which we can use to define the economic model.

SQ2 What are the requirements of a platform that supports implementing these methods?

Rationale: The requirements of the system are the basis for the design (SQ3), deployment (SQ4) and evaluation (SQ5).

SQ3 What are the main components of such platform and how do they interact with their target environment?

Rationale: Answering SQ3 is necessary to establish the functional design of the system.

SQ4 How can the platform be put into operation?

Rationale: Due to the distributed nature of the system, the deployment is non-trivial and should be addressed separately in more detail.

SQ5 Does the designed platform meet its requirements?

Rationale: The evaluation of the system is required 1) as input for further research, and 2) to assess the "utility, quality, and efficacy of the IT artifact [13]."

2.2 Research Methods

The following methods are used in order find an answer to the research questions.

2.2.1 Literature Review

In order to 1) find examples of existing tools that serve similar goals and 2) acquire a deeper understanding of the problem space, we review existing literature using the snowballing technique [31].

First, some knowledge in the area of *socioeconomics* is necessary to understand how participants would react to various types of reward systems. Second, since the main output of this research is the architecture of a software system, we review literature in the field of *software architecture* to obtain knowledge on definitions and standard methods. Finally, an important aspect of the architecture is the distributed nature of the platform and how to share state between arbitrary participants. An appropriate architectural pattern applied in such contexts is a *distributed ledger*, which we therefore discuss in more detail.

The results of the literature review are discussed in Chapter 3.

2.2.2 Design Science

In order to develop the architecture for the micro economy platform, the Design Science Research method [13] is applied. This method is appropriate given that the envisioned end result is an IT artifact which addresses a currently unsolved problem.

The application of design science in our research is structured around the design cycle as presented in the framework by Wieringa [30], and consists of the following five tasks:

- 1. Problem investigation: explore the context of the artifact
- 2. Treatment design: define/refine the design of the artifact and its specification
- 3. Treatment validation: check whether the designed artifact meets its requirements
- 4. Treatment implementation: apply the artifact in original problem context
- 5. Implementation evaluation: investigate how the artifact responds in its real-world context

The five tasks roughly follow the sub-questions as posed in Section 2.1, where we begin by exploring the context through literature and studying the case subject (SQ1). Subsequently, the *treatment design* and *validation* are specified iteratively through defining the platform's requirements (SQ2, Chapter 4) and architecture definition (SQ3, Chapter 5). Then, the *treatment implementation* is performed by developing a proof of concept of the architecture and testing it in the context of TTN (SQ4, Chapter 6), and finally, we *evaluate* the implementation through a case study (SQ5, also Chapter 6).

2.2.3 Case Study

The case study performed as part of the evaluation follows the research method as described by Yin [33]. The case study entails implementing a proof of concept within the context of The Things Network to validate the architecture. Additionally, we observe to what extent it supports the initiative in developing an economic model. The implementation is subsequently analyzed, both dynamically and statically. We measure the runtime performance of the proof of concept in a real world setting to spot potential scalability issues in the design, and assess any vulnerabilities from the perspective of security. The results are discussed in Chapter 6.

2.3 Contributions

2.3.1 Scientific

The main scientific contribution of this work is the addition of an architecture of a distributed micro economy platform to the body of knowledge on Software Architecture. To the best of the authors' knowledge, such system did not yet exist at the time of writing. Additionally, while there has been a lot of discussion around the fundamental theory of distributed ledgers, little literature is available on actual applications of this technology. The challenges encountered while applying the technology yield insights that add to the body of knowledge of distributed ledgers.

2.3.2 Societal

While in our research we specifically explore the case of an IoT initiative, the problem that it addresses can be generalized and is applicable to any DII. This is especially relevant in the current age where we observe a trend of an increasing number of peer-to-peer sharing economy initiatives [12]. However, where examples such as Uber, Etsy and AirBnB rely on a central for-profit organization, the platform proposed in this research contributes to a more decentralized structure for DIIs.

Chapter 3

Theoretical Background

3.1 Socio-Economic Perspective

In the introduction, we stated the assumption that the growth of the current model is limited due to the lack of incentives for participants to contribute to the infrastructure. In this section, we aim to provide supporting evidence for this claim and possible solutions through examples from literature.

3.1.1 Economic Principles

The actions of a collective consist of the actions of its individual agents. Therefore, to analyze how the ecosystem acts as a whole, we have to analyze individual behavior [19]. Following the Rational Choice Theory, individuals are generally self-interested and tend to choose those actions that benefit themselves the most [23]. When we apply these principles to the context of a DII, we see that if participants can get away with using the infrastructure without contributing to it, they generally will. Nevertheless, if participants gain something from contributing, e.g. earnings, learning or any other benefit, they have a direct incentive to contribute and are expected to be more inclined to do so. However, not every participant is capable of providing the infrastructure services themselves, since it requires both technical expertise and capital investments. So, given that 1) the pool of participants that are able to contribute is limited, and 2) not everyone from this pool has the proper incentives to contribute beyond their own needs, we believe that there is a significant risk the DII will remain limited in size if the underlying economic model is not changed.

To overcome this limitation, we can learn from traditional free markets. There, as long as it is possible to make a profit by providing a service, it is highly likely that at some point an entrepreneur will provide that service. However, a DII does not have any built-in mechanisms that enable providers to profit, at least not in the case of the example we introduced in Section 1.1. So, if we want to incentivize value adding activities, we should consider making them profitable. Note that *profit* in this context does not need to refer to *monetary* profit. As long as the incentive is large enough to trigger participants to contribute, it satisfies our goal.

3.1.2 Traditional Buyer-Seller Market

A traditional approach would be to enable providers to charge their users, e.g. in the form of a monthly subscription fee. However, there are various reasons we think this would not be suitable in the context of DIIs. First, due to the decentralized setting, users directly interact with many suppliers and vice versa, so it is difficult to establish a buyer-seller relationship and agree on predetermined fees. Second, the current vision of the example DII entails offering the infrastructure free of charge, only depending on voluntary contributions. While within this vision, there is room for commercialization, the competition of the free infrastructure will naturally reduce the profitability of commercial offerings.

3.1.3 Mining Rewards

In recent years, other approaches have also been employed, especially in the emerging domain of blockchain technology [26]. The primary example of an economic model that is completely different, is that of Bitcoin [18]: a decentralized peer to peer currency. Bitcoin itself could also be regarded as a DII, where the output of the initiative is an infrastructure for payments. In order to keep the infrastructure running, the initiative relies on part of the community to engage in an activity called *mining*. Mining is a resource intensive task which requires capital investments. In order to incentivize users to do these investments, the network periodically distributes a reward among those who contributed. The reward takes the form of an amount of the native currency, Bitcoin, and consists of a fixed number plus the transaction fees paid for by the users to conduct their payments. Over time, the fixed reward gradually decreases to zero. Therefore, the economic model is deflationary, which increases the incentive to *mine*, because following the laws of supply and demand, the value of the currency will go up when faced with an increase in demand.

3.1.4 Distributed Peer-to-Peer Marketplace

Another way to incentivize users to contribute to a shared infrastructure, is introduced by Swarm [29], a distributed storage platform based on Ethereum blockchain technology. The goal of Swarm is to provide peer-to-peer serverless storage hosting, and so it fully depends on its participants to operate the infrastructure. The incentive layer that has been built into Swarm serves to keep the balance of the services exchanged between participants in check. It accomplishes this through *pair-wise accounting* with negotiable prices. That is, each pair of peers that engages in trading services negotiates a price upfront, and each node maintains a list of offered and received services with its peers. As soon as the balance either surpasses the preconfigured payment or disconnect threshold, a payment is triggered or the service is discontinued. To summarize, the approach introduced by Swarm combines elements of a traditional market with buyers and sellers with a digital currency.

3.2 Software Architecture

One of the main outputs of this research is an *architecture* for a software system. To better understand what architecture is and how we should address it, the following section explores software architecture from a theoretical perspective.

3.2.1 Definitions

Architecture

According to [14], architecture is defined as the "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution". This definition is still very abstract, but its core aspects are discussed in more detail in [22].

"Fundamental properties" tell us *what* a system does (externally visible behavior) and *how* it does it (quality properties). The externally visible behavior defines the functional interactions between the system and its environment, whereas quality properties express non-functional aspects such as performance, maintainability, security, etc.

"Elements and relationships" refer to the pieces the system consists of and their relationships, in other words, the *structure*. This can be further decomposed into *static* and *dynamic* structures. *Static* structures relate to the *designtime* arrangement of elements, e.g. a module hierarchy, whereas *dynamic* structures indicate the *run-time* arrangement of elements, which could for example be expressed in a sequence diagram.

Finally, *principles of design and evolution* are defined as "fundamental statements of beliefs, approaches, or intents that guide the definition of your architecture". In practice, this boils down to practical guidelines architects use throughout a project to base design decisions on. For example, a design principle such as "prefer security over performance" would provide a guideline to architects whenever they have to make a decision with a trade-off between those two quality concerns.

Architecture description

Note that every system has an architecture, but not every architecture is documented. So, the architecture is an implicit concept, and an *architectural description* (AD) is used to make the architecture of a system explicit. The AD consists of a set of products such as models that document the architecture. In order to provide a comprehensive description, the AD discusses the architecture from multiple angles, or *views*.

A view is defined as a "representation of one or more structural aspects of an architecture that illustrates how the architecture addresses one or more concerns held by one or more of its stakeholders." Here, *stakeholders* represent the individuals for whom the system is built.

3.2.2 Viewpoints and Perspectives

To standardize the way views are represented in an AD, [22] introduces the notion of *viewpoints* and *perspectives*. Viewpoints are collections of patterns, templates and conventions to describe structural aspects. Additionally, perspectives are similar to viewpoints in that they provide patterns, templates and conventions, but are concerned with quality requirements of a system related to aspects such as security, maintainability or performance. Since they cannot be attributed to a single viewpoint, they are discussed separately.

Figure 3.1 depicts the views that are defined in [22] and how these are related. Briefly summarized, the *context view* describes the system as a black box in its environment (people, systems and external entities), and is useful to indicate the responsibilities of the system. It also defines the scope, context and interfaces for the *software design*, which is described in three separate viewpoints, each addressing different concerns.

First, the *functional view* describes what the system does, i.e. the elements that deliver its functionality and how they are related. Second, the *information view* describes how the system stores, manipulates, manages and distributes information, and third, the *concurrency view* describes what parts of the system execute concurrently and how this is coordinated.

The *development view* describes how the software development process is organized. This includes both the source code organization as well as the surrounding processes such as design and testing.

Finally, the *deployment* and *operational* views describe the system in its live environment. The deployment view prescribes the runtime environment and addresses concerns such as the network, hardware and other platform requirements, whereas the operational view provides a description on how the system will be controlled, managed and monitored while running. This includes concerns such as configuration management, alerting, migrations, etc.

Additionally, we can distinguish various *perspectives* that describe crosscutting concerns. For example, the *security perspective* discusses who can control and access which parts of a system and the *performance perspective* is concerned with aspects such as throughput and latency.

An AD typically does not include all of the previously described viewpoints and perspectives, but rather focuses on those views that are most relevant to the system. For example, a simple single-tier system deployed on a single machine without any special requirements can omit the deployment view. However, the same view is essential to understand the runtime environment of a heavily distributed system.



Figure 3.1: Views and their relationships as described in [22]

3.3 Distributed Ledgers

Distributed ledgers are an emergent topic and have received a lot of attention recently due to the popularization of the concept of *blockchains*. Essentially, a distributed ledger is a distributed database which enables consensus between parties that do not trust each other [26]. The database consists of an append-only sequence of transactions and therefore, 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, for example digital currencies.

A blockchain is a specific type of distributed ledger, where transactions are grouped in a sequence of blocks and each block references the previous block. 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 [18], or to the actual instantiation of the pattern as applied in projects such as Bitcoin and Ethereum [32]. For the purpose of this research, we are mainly interested in the generic architectural pattern of a distributed ledger, to assess to what extent it can solve problems we encounter in designing the micro economy platform.

In the remainder of this section we review different aspects of distributed ledgers. Over the previous years, many different implementations have been developed, and while they all aim to solve the trustless distributed consensus problem, the actual application and implementation details vary substantially.

3.3.1 Public vs Private vs Hybrid

One aspect that varies across distributed ledger implementations is to what extent it is open and decentralized. We can distinguish three categories: public, private and hybrid [15]. A public ledger is fully decentralized and open. It assumes that there is no trust between any of the participants, which requires a fully distributed consensus mechanism where any member of the network can validate transactions. A hybrid ledger (or public-permissioned) is still open for anyone to join, but instead of allowing anyone to validate transactions, it depends on a consortium of trusted parties. Finally, the most restrictive form is a private ledger, which is implemented within an organization and only allows that organization to submit transactions. In this case, one might argue that the main benefit of a distributed ledger, namely that of achieving consensus in trustless settings, is lost.

3.3.2 Smart Contracts

Another aspect that differs from ledger to ledger is whether they support *smart* contracts. While popularized by Ethereum [5], Smart Contracts were actually first defined in [27]. 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.

For example, given that a project needs a capital injection, it can kick off a crowdsale smart contract, which accepts payments from arbitrary donators. As

soon as a preconfigured time limit or threshold of payments has been exceeded, the smart contract can close and trigger the subsequent actions. If the total of payments did not exceed the threshold in time, the payments of donators that did contribute could automatically be returned for example. In essence, the smart contract here keeps track of some state and is accompanied by arbitrary logic to manipulate that state.

In Ethereum, smart contracts are executed as part of transactions on the Ethereum Virtual Machine (EVM), a quasi-Turing-complete virtual state machine [32, 5]. 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.

3.3.3 Consensus Mechanisms

Due to the decentralized nature of distributed ledger networks, they need some mechanism to reach consensus on the current state of the blockchain. Over the past years, different approaches have been explored, which are summarized in the following subsections.

Proof-of-Work

The first blockchain introduced to the general public, Bitcoin, uses a *Proof-of-Work* mechanism [18] which was inspired by a hypothetic e-mail spam prevention technique first introduced in 1993 [9]. By requiring senders of e-mail to solve a computationally intensive puzzle and provide the answer in the header of the e-mail, it becomes economically infeasible to send spam e-mails. Recipients can subsequently verify that the sender has invested some resource into sending the e-mail, which reduces the likelihood of that particular e-mail being spam.

In the context of blockchains, the Proof-of-Work consensus mechanism works as follows:

- 1. Nodes broadcast new transactions to all other nodes in the network;
- 2. Every node works on finding a solution to a puzzle that takes the current set of pending transactions and a reference to the previous block as input;
- 3. The first node to find the solution broadcasts the newly discovered block consisting of the solution and transactions to the rest of the nodes;
- 4. Other nodes either accept or reject the block depending on whether it is correct, i.e. the hash and all transactions are valid;
- 5. Go back to 1.

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.

Proof-of-Space

One of the main drawbacks of the Proof-of-Work approach is that the computations require a significant amount of energy resources. Some researchers estimate that a Proof-of-Work network at scale would incur a 2.1% increase in carbon dioxide emissions worldwide [2]. In an effort to put the invested resources by the network to better use, several attempts [10, 17, 20] have been made to base a digital currency on network participants offering disk space instead of computational power. Instead of solving a cryptographical puzzle, participants store data. To verify they indeed store the data, the network issues random challenges that require the storer to submit a proof that it in fact still can access the original files.

Proof-of-Stake

While Proof-of-Space indeed does not use computational power, it still depletes a physical resource, namely disk space. Therefore, in [4] another alternative consensus mechanism that does not rely on spending computational power is proposed: Proof-of-Stake. Where in Proof-of-Work and Proof-of-Space the probability that a user can create a new block is relative to their depletion of a physical resource, i.e. computational power or disk space, in Proof-of-Stake, this probability is relative to the *stake* users have in the system, i.e. the amount of currency they have in the ecosystem. By burning the stake when a user provides invalid blocks, the system provides an incentive to validate blocks in an honest manner.

Proof-of-Authority

The final consensus mechanism we found in literature, Proof-of-Authority [8], is only partially decentralized and therefore mainly suitable for private or semiprivate blockchains. Instead of relying on an anonymous third party group of validators, in Proof-of-Authority, a consortium of trusted parties is privileged to create new blocks. These trusted parties are generally larger established corporations such as banks or telecom operators. While it is less decentralized than the previously discussed consensus mechanisms, it offers a conceptually simpler model which is easier to maintain. Therefore, if pure decentralization is not a must, Proof-of-Authority can prove to be a good alternative.

3.3.4 Blockchain-Free Distributed Ledgers

Note that 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 [16, 6, 21] 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 [21]. 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 Proof-of-Work that includes data from the earlier transactions. By providing the Proof-of-Work 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 Proof-of-Work 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.

Chapter 4

Requirements for a Micro Economy Platform

4.1 Purpose & Scope

The main purpose of the micro economy platform proposed in this research is to enable distributed infrastructure initiatives 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. The details of these services and the reward scheme are specific for each DII and should therefore be freely configurable.

4.2 Stakeholders

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. Firstly, *contributors* are participants who add value by contributing to the infrastructure, e.g. the *gateway operators* and *routing service providers* discussed in Section 1.1. Secondly, *users* are participants that utilize the infrastructure, e.g. the *application owners* who deploy applications on the IoT network in the context of TTN. These two types are not mutually exclusive, i.e. a contributor can simultaneously be 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.

Example 4.1 If we apply these roles to the example as introduced in Section 1.2, Alice and Netcorp are contributors, since they provide gateways and routing services. Parco is a user, since they utilize the network infrastructure for their parking garage application, and Bob finally is an investor

who speculates on the value of the currency introduced by the DII.

4.3 Functional Requirements

The following section describes the basic functional requirements for each of the main use cases of the micro economy platform.

4.3.1 Reward Model

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

Example 4.2 Alice has one gateway which in the preceding hour has handled 600 messages with a total airtime – the time the gateway radio is active – of 120 seconds, and an uptime of 100%. To calculate the current score of Alice's gateway, we execute the configured *score–function* and pass in the following set of metrics: $\{600, 120, 100\}$. The function is currently defined as follows:

 $score(X) = \begin{cases} 5 & \text{if } x_{airtime} \in [100s, \infty) \\ 2 & \text{if } x_{airtime} \in [1s, 100s) \\ 1 & \text{if } x_{airtime} \in [0.01s, 1s) \\ 0 & \text{otherwise} \end{cases}$

Here, we define several tiers a gateway can be assigned to based on the processed airtime. An initial threshold of 0.01s is introduced to prevent inactive gateways from receiving a reward.

Since Alice's gateway is in the upper tier, the gateway receives a score of 5 for this period. If the moving average window is 5 periods, and the scores in the previous 4 periods were 0, 1, 2 and 2, then the current score of the gateway is $\frac{0+1+2+2+5}{5} = 2$. If Alice has another gateway with score 1, then Alice's overall Karma score is 3 at that point in time.

Note that the scoring function definition above is slightly arbitrary and should be tuned over time based on feedback gathered from actual operations.

For example, it might be desirable to include the number of messages or uptime metrics in future versions of the scoring function. In case the scoring function is changed, only scores from that moment onwards will be calculated using the new function, i.e. the system does not need to retroactively compute scores.

4.3.2 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

Example 4.3 Consider the following scenario. The reward is set to 250 Wavelets per interval, and at the current interval the following Karma scores have been computed: Alice: 7, Netcorp: 2, and TTN: 1. Then, because their total Karma combined is 10, Alice would receive 70% of the reward, Netcorp 20% and TTN 10%. Since the reward is set to 250, Alice receives 175, Netcorp 50, and TTN 25 Wavelets.

4.3.3 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.

Example 4.4 Alice has a gateway that is consistently handling traffic from both her own and the Parco applications, and therefore has a positive Karma score. In order to capitalize on the positive Karma score and start receiving Wavelets, she has to let the platform know where to send the tokens, i.e. her wallet address. This requires her to first obtain a wallet, which entails using the wallet software to generate a wallet, and subsequently follow an authentication flow to prove she indeed controls that particular TTN account. After this set-up, her wallet address is registered to the user account on the blockchain and she will start receiving rewards

4.3.4 Send/Receive 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

Example 4.5 Alice has an account balance of 300 wavelets and wants to send Bob 100 Wavelets, whose account is empty. She submits a transaction of 100 Wavelets to Bob's wallet. After the transaction has been verified, Alice has 200 Wavelets, and Bob has 100 wavelets.

4.3.5 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 5.2.3. 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

Example 4.6 Parco wants to continue rolling out their IoT application, but struggle with the fact that they do not have any guarantees that the network will be maintained properly. They discover through the market-place that Netcorp offers an SLA on gateways that cover their area of operation. In exchange for 5 wavelets per hour, they guarantee an uptime of 99.99% per month. Parco subscribes to the SLA, and deposits sufficient funds to pay for a few weeks of service. On an hourly basis, the smart contract checks the metrics of the concerned infrastructure components. If the conditions of the SLA hold, the smart contract pays out the hourly fee. Otherwise, a penalty clause is triggered, which in this case consists of burning a deposit Netcorp put upfront. However, the latter does not occur, and after a few weeks, Parco is still satisfied with the service and continue resubscribing to the service.

4.4 Non-Functional Requirements

Orthogonal to the functional requirements, we list several non-functional requirements below that do not prescribe *what* the platform should do, but *how* it should perform.

4.4.1 Efficient Batch Handling

As described in Sections 4.3.1 and 4.3.2, the platform processes metrics in batches. The total running time of a naive implementation of these batches would grow linearly with an increasing number of infrastructure components. Given that the running time of a single batch should not exceed the interval between batches, the maximum capacity of the platform is limited by the efficiency of the batches. While the current number of infrastructure components in the case subject is relatively small in the order of magnitude of several thousands, the platform should be able to scale towards hundreds of thousands of components.

4.4.2 Distributed Deployment

One of the strengths of distributed infrastructure initiatives 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.

4.4.3 Security

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

Chapter 5

Micro Economy Platform Architecture

The following chapter describes the architecture of the micro economy platform, based on the requirements specified in the previous chapter. The architecture description follows the viewpoint catalog of [22], which is described in more detail in Section 3.2.2.

5.1 Context Viewpoint

5.1.1 Context Diagram

Figure 5.1 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 5.2.3. Finally, a third group of outside actors, investors, might exchange tokens with participants without actively participating in the network.

Example 5.1 Figure 5.2 gives a concrete example of the platform context when implemented at TTN. The main interaction between the technical infrastructure and platform consists of submitting operational metrics, based on which the contributions are determined. Additionally, to be able to link user accounts from the infrastructure to accounts in the micro economy platform, the infrastructure provides user details through one or more identity providers.



Figure 5.1: Context diagram of micro economy platform



Figure 5.2: Context diagram of micro economy platform applied to TTN

5.1.2 Economic Model



Figure 5.3: Economic model as applied in the micro economy platform

The economic model employed by the micro economy platform is summarized in Figure 5.3. 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 an hourly basis, a fixed number of tokens is created and subsequently divided over all contributors, proportional to the size of their contributions. Contributions are quantified using a Karma score, as described in more detail in Section 4.3.1. This process is analogous to mining in Proof of Work-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 5.2.3. 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.

5.2 Functional Viewpoint

5.2.1 High-Level Structure

Figure 5.4 depicts the high-level platform design 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 temporarily stores the metrics. 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, because 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*.

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* as described in Section 4.3.2.

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 as described in Section 4.3.3.

Finally, participants can use their tokens to purchase services from other participants in a *marketplace*. The services are user defined, but an example of a possible service is provided in Section 5.2.3.



Figure 5.4: High-level platform structure

5.2.2 Token Reward Workflow

Figure 5.5 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.

5.2.3 User Defined SLA Workflow

Figure 5.6 describes how a user defined SLA could work in practice. Note that it would ultimately be up to users themselves to define them, 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 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.



Figure 5.5: Token reward workflow



Figure 5.6: User defined SLA workflow

5.3 Deployment Viewpoint

5.3.1 Runtime Environment

Figure 5.7 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 discussed in Section 5.2.1 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 we can fully trust. 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*.

5.3.2 Technology Dependencies

The architecture is agnostic of specific technologies. However, the architecture does make various assumptions about the functionality and characteristics of the third party software it depends on.

First, the architecture assumes the availability of a *blockchain* that supports programmable *smart contracts*. For example, the Karma-contract must be able to make arbitrary calculations to convert infrastructure metrics to Karma scores as described in 4.3.1. Additionally, the blockchain software should support a hybrid deployment, given that 1) the cost of using a public blockchain would be out of proportion, and 2) a private deployment does not allow arbitrary third parties to participate. Since the platform will be deployed in a hybrid setting, it should also support a consensus mechanism such as Proof of Authority (cf.



Figure 5.7: Runtime environment

Section 3.3.3), which does not require validators to deplete a significant amount of resources as Proof of Work does. An example of a blockchain that satisfies these requirements is Ethereum.

Second, the monitor component depends on an *event store* that provides the capability to temporarily persist metrics and aggregate these metrics over specific periods of time. An ideal candidate would be a time-series database such as InfluxDB, since it has built-in support for aggregation over time and also provides mechanisms to automatically discard old data.

Finally, the *identity provider* must provide some authentication mechanism, e.g. OAuth, to facilitate linking infrastructure users to blockchain accounts. This is necessary to map contributions in the infrastructure to the right wallets on the blockchain.

Chapter 6

Evaluation: Case Study at The Things Network

6.1 **Proof-of-Concept Implementation**

In order to validate the design of the architecture and provide a starting point for a real-world implementation, we developed a proof of concept of the micro economy platform. The proof of concept 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.

Figure 6.1 provides an overview of which aspects have been implemented and which have been omitted. In essence, the proof of concept 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.



Figure 6.1: Overview of implemented components. Note that only the shaded components with bold text (e.g. Monitor) have been implemented.

In the remainder of this section, the implemented components are discussed in more detail.

6.1.1 Integration Agent

For the proof of concept, 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.

Monitor



Figure 6.2: Monitor implementation

Figure 6.2 depicts a more detailed view of the implemented monitor. Given that in the context of TTN infrastructure components already expose data over gRPC¹ streams and bindings for these streams are available in the Go programming language², 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 InfluxDB³, 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.

For the proof of concept, we only implemented the collection of *uplink* and *downlink* events (see Section 1.1). For these events, the amount of airtime is stored per message as a measurement. Since airtime indicates the amount of time a radio is active transmitting or receiving, it provides a metric for the utilization of a device. The set of metrics is deliberately kept simple for the proof of concept, but the goal is to keep track of more sophisticated metrics in the future, such as component uptime or signal strength. Currently, only metrics about gateways are tracked, but the goal is that eventually also metrics about other infrastructure components such as routers, brokers and handlers are collected.

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 over the previous period through a query on the InfluxDB data store.

¹See https://grpc.io/about/

²See https://golang.org/

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

Example 6.1 Imagine Alice has a gateway that has handled 2 uplink messages of 50ms each and 1 downlink message of 100ms in a given period, then the metrics submitted for that particular gateway would be [2, 50, 1, 100]. Submitting the metrics occurs by sending a transaction to the smart contract on the blockchain that keeps track of the metrics.

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 proof of concept.

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

6.1.2 Blockchain

Most of the core functionality has been implemented through smart contracts on the private blockchain. For the proof of concept, there are two concepts of importance: 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 proof of concept 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,



Figure 6.3: Sequence diagram of wallet registration

other languages could be developed to build smart contracts for EVM-enabled blockchains. Listing 6.1.2 shows a simple example of a smart contract in Solidity that keeps track of an arbitrary token and the balances of its users.

As visible from this example, 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.

Note that instead of using the public Ethereum chain, the proof of concept 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.

Implemented Smart Contracts

Figure 6.4 depicts the actually implemented smart contracts as a simplified UML class diagram.

Note that although the architecture suggest a separation between the Karma and Infrastructure Metrics Store contracts (cf Figure 6.1), the functionality of both contracts has been implemented in the Karma contract for the purpose of this proof of concept. 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 5.2.2 and of

```
contract Token {
    address owner:
    mapping(address => uint) balances;
    function Token() {
          'msg.sender'
                      contains the address of the 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:
      ł
      balances[msg.sender] -= amount;
      balances[to] += amount;
    }
}
```

Listing 1: Example of smart contract in Solidity

which an example is given in Section 4.3.1, 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. To that end, we define a Karma Scorer interface, which takes as input an arbitrary set of metrics and outputs a single Karma score. For the proof of concept, only a GatewayScorer has been implemented, but 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.

For more details on the actual implementation of the smart contracts, please



Figure 6.4: Overview of smart contracts and their relationships

refer to Appendix A which contains their source code.

6.2 Analysis

6.2.1 Scalability

The implementation of the proof of concept shows that the scalability 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 proof of concept 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. See Figure 5.5 for 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. Please refer to Appendix B for the details of the measurements.

Given that the interval between batches is one hour, there is not a lot of headroom to scale up in terms of number of tracked infrastructure components.

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 [24] 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 [11], 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 [3]. 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.

6.2.2 Security

Given that rewards contributors receive constitute some 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. There are several ways one could gain such advantage.

First, one could try to directly steal the rewards of another participant by gaining control of his/her private key. While it is certainly not impossible, this scenario is unlikely given that we use the standard Ethereum software which provides sufficient measures for users to protect themselves against such attacks. For example, the wallet is only accessible through a private key, which in turn is secured by a passphrase. Without the combination of the two, it is impossible to access the funds in a wallet, so as long as users keep these safe, there funds are too.

A second type of vulnerability could exist in the integration with the existing identity providers. If a user were to gain control over the account of a user in the existing infrastructure, it would be possible to generate a new wallet and link it to the other user's account, thereby receiving his/her rewards. However, here we again rely on standard software and protocols (e.g. OAuth) which have been peer reviewed by many experts, so any vulnerabilities are less likely.

Thirdly, the smart contracts could contain unforeseen vulnerabilities. Although the current smart contracts are relatively simple, it is possible that over time when they grow more complex vulnerabilities are introduced. Additionally, the underlying language and Ethereum Virtual Machine are prone to contain vulnerabilities as noted in [1]. We expect that this is in part due to immaturity in the technology that will solve itself over time.

Finally, a user could generate fake data and provide it to the monitor. For example, a user could implement a software gateway that generates a lot of messages. Without any additional mechanisms, the platform would pick this up, 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. First, the solution is not watertight. As soon as a component is "in", it is trusted and it can start generating fake data. It can be very hard (if not impossible) to detect the authenticity of the data, so this is a severe issue. 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 we are yet to come up with such solution.

Chapter 7

Discussion

7.1 Findings

Theory on distributed ledgers is advancing at a rapid pace, and an increasing amount of research is performed on that topic. However, the technology is still awaiting widespread adoption and real "killer applications", which is reflected in the literature on distributed ledgers. Most publications are concerned with fundamental aspects, or trying to provide an overview of the various concepts through taxonomies and ontologies. These are important topics, but exploring real-world applications is also essential to close the feedback loop and gain a deeper understanding of the potential and limitations of the technology.

To illustrate this, in the early phases of this research we regarded blockchain mainly as a means, i.e. a solution to the problem of achieving consensus in distributed systems in trustless contexts. However, as the project progressed, new capabilities became apparent that were only possible *because* of the use of a blockchain. For example, it was only after several iterations before we began to see the possibility to enable participants engage in SLAs that are self-enforced based on actual network metrics. We therefore found that without diving deep in an actual implementation, it is very difficult to foresee the full potential of a new technology.

7.1.1 Scalability

Nevertheless, during the implementation of these smart contracts we quickly ran into the current technical limits of blockchain, especially in the area of scalability. The proof of concept developed during the case study showed us that both computational and storage requirements quickly become too high to still 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.

7.1.2 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 that still does not provide any watertight proof on the truthfulness.

7.1.3 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 Proof of Work. 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.

7.2 Threats To Validity

The main limitations for these findings stem from the relative short timespan that was available for the research.

First, 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. We can therefore not make any claims on the true generalizability of the proposed platform. Second, the implemented proof of concept 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. Third, 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. Finally, the time between launching the platform in production and the evaluation was relatively short. We therefore expect that more lessons are still to be learned when the platform has been in production for a longer timespan.

On a more conceptual level, another limitation stems from the fact that the terms *distributed infrastructure initiative* and *micro economy* have been coined specifically for the purpose of this research, and do not originate from earlier

research. This was necessary since the authors could not find any fitting terms at the time of writing, but that does limit the rigor of the research.

Finally, the topic of this research crosses many different domains, from sociology to computer science. However, the technical background of the researchers may have introduced bias in the results, despite frequent discussions with experts from other domains.

Chapter 8

Conclusions

In the preceding chapters, we have described the development of an architecture for a platform that enables distributed infrastructure initiatives to develop their own micro economy. To conclude our research, in this chapter we reflect on the original research questions and address areas for future research.

8.1 Results

The original research question — What is an effective architecture for a platform that enables distributed infrastructure initiatives to develop their own micro economy? — has been addressed by answering the following sub-questions.

SQ1 What are appropriate methods to incentivize value adding activities? Literature provides little consensus on what the best methods are to incentivize people to conduct specific behavior. This is only natural given the complexity of human nature and the wide array of settings in which various experiments have been performed. For this research, we therefore specifically looked for approaches that have been applied in settings similar to that of TTN, i.e. decentralized communities mainly consisting of technology-oriented people. From the examples we found, the approach adopted by digital currencies such as Bitcoin, where contributors are rewarded by issuing an amount of freely exchangeable tokens proportional to their contribution, proved to be the most compelling. Firstly because their goals are well-aligned with those in our research, i.e. incentivize contributions to shared infrastructure, and secondly because this approach has been validated in practice on a large scale.

SQ2 What are the requirements of a platform that supports implementing a reward system?

The requirements of the platform are described in Chapter 4, and center around providing a platform to introduce a digital currency in the ecosystem of a distributed infrastructure initiative, and where new tokens of the currency are issued based on a configurable reward model similar to that of Bitcoin and Ethereum.

SQ3 What are the main components of such platform and how do they interact with their target environment? The answer to this research question is provided through the architecture for the micro economy platform as described in Chapter 5. The main components of the architecture are 1) a blockchain which hosts a set of smart contracts to introduce a digital currency and an economic model, and 2) a set of components to integrate the currency in an existing ecosystem.

SQ4 How can the platform be put into operation?

In order to validate the architecture, we implemented the core aspects in a proof of concept and deployed it in the context of The Things Network, a real-world example of a distributed infrastructure initiative. The results of the implementation are discussed in more detail in Chapter 6.

 ${f SQ5}$ Does the designed platform meet its requirements?

While the implementation of the proof of concept fulfills the functional requirements, there is still more research to be performed to address some concerns related to scalability and security. In essence, the current blockchain implementation does not provide sufficient performance to be scaled up significantly, and needs fundamentally more secure mechanisms to obtain metrics about the infrastructure components operations.

8.2 Future Research

The results of our research provide an initial architecture for a Micro Economy Platform for Distributed Infrastructure Initiatives, but the implementation has revealed several areas that require more research before it can be deployed at a larger scale.

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 [3]) to reduce the storage requirements, and 2) explore the possibility of using *payment channels* [7] for smart contracts to reduce the number of required transactions and thereby increasing the overall throughput.

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

- Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. "A Survey of Attacks on Ethereum Smart Contracts (SoK)". In: Principles of Security and Trust: 6th International Conference, POST 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings. Ed. by Matteo Maffei and Mark Ryan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 164–186. ISBN: 978-3-662-54455-6. DOI: 10.1007/978-3-662-54455-6_8.
- [2] Jörg Becker et al. "Can We Afford Integrity by Proof-of-Work? Scenarios Inspired by the Bitcoin Currency". In: *The Economics of Information Security and Privacy*. Ed. by Rainer Böhme. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 135–156. ISBN: 978-3-642-39498-0. DOI: 10. 1007/978-3-642-39498-0_7.
- Juan Benet. "IPFS Content Addressed, Versioned, P2P File System". In: CoRR abs/1407.3561 (2014). URL: http://arxiv.org/abs/1407.3561.
- [4] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. "Cryptocurrencies Without Proof of Work". In: Financial Cryptography and Data Security: FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers. Ed. by Jeremy Clark et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 142–157. ISBN: 978-3-662-53357-4. DOI: 10.1007/978-3-662-53357-4_10.
- [5] Vitalik Buterin. A Next-Generation Smart Contract and Decentralized Application Platform. 2014. URL: https://www.ethereum.org/pdfs/ EthereumWhitePaper.pdf.
- [6] A Churyumov. Byteball: a decentralized system for transfer of value. 2015. URL: https://byteball.org/Byteball.pdf.
- [7] Christian Decker and Roger Wattenhofer. "A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels". In: Stabilization, Safety, and Security of Distributed Systems: 17th International Symposium, SSS 2015, Edmonton, AB, Canada, August 18-21, 2015, Proceedings. Ed. by Andrzej Pelc and Alexander A. Schwarzmann. Cham: Springer International Publishing, 2015, pp. 3–18. ISBN: 978-3-319-21741-3. DOI: 10.1007/978-3-319-21741-3_1.

- [8] Tien Tuan Anh Dinh et al. "BLOCKBENCH: A Framework for Analyzing Private Blockchains". In: Proceedings of the 2017 ACM International Conference on Management of Data. SIGMOD '17. Chicago, Illinois, USA: ACM, 2017, pp. 1085–1100. ISBN: 978-1-4503-4197-4. DOI: 10.1145/3035918.3064033. URL: http://doi.acm.org/10.1145/3035918.3064033.
- [9] Cynthia Dwork and Moni Naor. "Pricing via Processing or Combatting Junk Mail". In: Advances in Cryptology — CRYPTO' 92: 12th Annual International Cryptology Conference Santa Barbara, California, USA August 16–20, 1992 Proceedings. Ed. by Ernest F. Brickell. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 139–147. ISBN: 978-3-540-48071-6. DOI: doi:10.1007/3-540-48071-4_10.
- [10] Stefan Dziembowski et al. "Proofs of Space". In: Advances in Cryptology CRYPTO 2015: 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II. Ed. by Rosario Gennaro and Matthew Robshaw. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 585–605. ISBN: 978-3-662-48000-7. DOI: 10.1007/978-3-662-48000-7_29.
- [11] Davide Frey et al. "Bringing Secure Bitcoin Transactions to Your Smartphone". In: Proceedings of the 15th International Workshop on Adaptive and Reflective Middleware. ARM 2016. Trento, Italy: ACM, 2016, 3:1-3:6. ISBN: 978-1-4503-4662-7. DOI: 10.1145/3008167.3008170. URL: http: //doi.acm.org/10.1145/3008167.3008170.
- [12] Juho Hamari, Mimmi Sjöklint, and Antti Ukkonen. "The sharing economy: Why people participate in collaborative consumption". In: *Journal* of the Association for Information Science and Technology 67.9 (2016), pp. 2047–2059.
- [13] Alan R Hevner et al. "Design science in information systems research". In: MIS quarterly 28.1 (2004), pp. 75–105.
- [14] "ISO/IEC/IEEE Systems and software engineering Architecture description". In: ISO/IEC/IEEE 42010:2011(E) (Dec. 2011), pp. 1–46. DOI: 10.1109/IEEESTD.2011.6129467.
- [15] Joost de Kruijff and Hans Weigand. "Understanding the Blockchain Using Enterprise Ontology". In: Advanced Information Systems Engineering: 29th International Conference, CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings. Ed. by Eric Dubois and Klaus Pohl. Cham: Springer International Publishing, 2017, pp. 29–43. ISBN: 978-3-319-59536-8. DOI: 10.1007/978-3-319-59536-8_3.
- [16] Sergio Demian Lerner. DagCoin: a cryptocurrency without blocks. 2015. URL: https://bitslog.files.wordpress.com/2015/09/dagcoinv41.pdf.
- [17] Andrew Miller et al. "Permacoin: Repurposing Bitcoin Work for Data Preservation". In: Proceedings of the IEEE Symposium on Security and Privacy. IEEE, May 2014. URL: https://www.microsoft.com/enus/research/publication/permacoin-repurposing-bitcoin-workfor-data-preservation/.

- [18] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system.* 2008. URL: https://bitcoin.org/bitcoin.pdf.
- [19] Mancur Olson. The Logic of Collective Action: Public Goods and the Theory of Groups. Harvard University Press, 1974.
- [20] Sunoo Park et al. Spacecoin: A cryptocurrency based on proofs of space. Tech. rep. IACR Cryptology ePrint Archive, 2015: 528, 2015.
- [21] Sergei Popov. The tangle. 2016. URL: https://iotatoken.com/IOTA%5C_ Whitepaper.pdf.
- [22] Nick Rozanski and Eóin Woods. Software systems architecture: working with stakeholders using viewpoints and perspectives. 2nd ed. Addison-Wesley, 2012.
- [23] John Scott. "Rational Choice Theory". In: Understanding Contemporary Society: Theories of the Present. 2000, pp. 126–138.
- [24] State Tree Pruning. URL: https://blog.ethereum.org/2015/06/26/ state-tree-pruning/.
- [25] Arun Sundararajan. The Power of Connection: Peer-to-Peer Businesses. 2014. URL: https://smallbusiness.house.gov/UploadedFiles/1-15-2014%5C_Revised%5C_Sundararajan%5C_Testimony.pdf.
- [26] Melanie Swan. Blockchain: Blueprint for a new economy. "O'Reilly Media, Inc.", 2015.
- [27] Nick Szabo. "Formalizing and securing relationships on public networks". In: *First Monday* 2.9 (1997).
- [28] The Things Network. URL: https://thethingsnetwork.org.
- [29] Viktor Trón et al. SWAP, SWEAR and SWINDLE: Incentive system for Swarm. May 2016. URL: http://swarm-gateways.net/bzz:/theswarm. eth/ethersphere/orange-papers/1/sw%5C%5E3.pdf.
- [30] Roel J Wieringa. Design Science Methodology for Information Systems and Software Engineering. Springer, 2014.
- [31] Claes Wohlin. "Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering". In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering. EASE '14. London, England, United Kingdom: ACM, 2014, 38:1–38:10. ISBN: 978-1-4503-2476-2. DOI: 10.1145/2601248.2601268. URL: http://doi.acm.org/10.1145/2601248.2601268.
- [32] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. 2014. URL: https://bravenewcoin.com/assets/Whitepapers/ Ethereum - A - Secure - Decentralised - Generalised - Transaction -Ledger-Yellow-Paper.pdf.
- [33] Robert K Yin. Case study research: Design and methods. Sage publications, 2003.

Appendix A

Smart Contracts

```
contract KarmaScorer {
1
^{2}
         function score(int64[] metrics) constant returns (int64);
3
     }
4
5
     contract GatewayScorer is KarmaScorer {
6
         struct Tier {
7
             int64 threshold;
 8
9
              int64 reward;
         }
10
11
         Tier[] tiers;
12
^{13}
14
         function GatewayScorer(int64[] thresholds, int64[] rewards) {
15
             if (thresholds.length != rewards.length) {
16
                  return;
              }
17
18
              tiers.length = thresholds.length;
for (uint i = 0; i < thresholds.length; i++) {</pre>
19
^{20}
^{21}
                   tiers[i].threshold = thresholds[i];
                   tiers[i].reward = rewards[i];
^{22}
              }
^{23}
         }
^{24}
^{25}
26
         function score(int64[] metrics) constant returns (int64)
27
         {
              if (metrics.length < 4) {
^{28}
^{29}
                  return 0;
              }
30
31
              int totalAirtime = metrics[1] + metrics[3];
32
33
              for (uint i = 0; i < tiers.length; i++) {
^{34}
                  if (totalAirtime >= tiers[i].threshold) {
                       return tiers[i].reward;
35
                  }
36
37
             }
38
39
              return 0;
40
         }
    }
41
```

```
contract Karma {
 1
          uint public constant blockReward = 250 * 10**18;
2
3
          struct KarmaPart {
4
              bytes32 sourceRef;
\mathbf{5}
 6
              int64 karma;
7
         }
 8
          struct BlockData {
9
10
              int64[] metrics;
11
              int64 score;
12
              int64 karma;
^{13}
              KarmaPart[] karmaParts;
^{14}
              bool mined;
         }
15
16
          struct BlockMeta {
17
18
              BlockState state;
19
              int64 totalKarma;
          3
20
21
          enum BlockState { New, Sealed, Mined }
22
23
^{24}
          ACL acl;
^{25}
          ComponentDirectory componentDir;
26
          UserDirectory userDir;
27
          Wavelets wavelets:
28
          mapping(uint => KarmaScorer) karmaScorers;
^{29}
30
          uint32 windowSize = 500;
^{31}
          uint32 public lastBlock;
32
         mapping(uint32 => mapping(bytes32 => BlockData)) public data;
mapping(uint32 => BlockMeta) public meta;
33
34
35
          function Karma(ACL _acl, ComponentDirectory _karmaDir, UserDirectory _userDir, Wavelets _wavelets) {
36
37
              acl = _acl;
38
              componentDir = _karmaDir;
              userDir = _userDir;
wavelets = _wavelets;
39
40
          }
41
42
          function update(uint32 blockNo, bytes32 componentRef, int64[] input) returns (bool) {
43
44
               // To prevent karma and wavelets being issued multiple times, check if metrics have been set before
45
              if (data[blockNo][componentRef].metrics.length > 0) { return; }
46
              if (meta[blockNo].state != BlockState.New) { return; }
47
^{48}
^{49}
               // Do not process karma sources without collaborators
50
              uint numCollaborators = componentDir.getComponentCollaboratorCount(componentRef);
              if (numCollaborators == 0) {
51
52
                   return;
              3
53
54
              // Retrieve the karma source type (e.g. gateway, router, etc.)
55
56
              uint sourceType = componentDir.getComponentType(componentRef);
57
              // Store the metrics for future reference
58
              data[blockNo][componentRef].metrics = input;
59
60
               // Use the source-type-specific scorer to calculate the karma score at the present
61
              data[blockNo][componentRef].score = karmaScorers[sourceType].score(input);
62
63
64
              // Calculate the current karma score, which is a moving average of the past fwindowSize scores
int64 karmaToRemove = data[blockNo-windowSize][componentRef].score / int64(windowSize);
65
              int64 karmaToAdd = (data[blockNo][componentRef].score / int64(windowSize));
66
67
              int64 currentKarma = data[blockNo-1][componentRef].karma;
68
69
               // In case of missing data points, it can occur that the moving average drops below 0.
              // During normal operation this should not happen, but if it does, we reset the average.
70
              if (currentKarma - karmaToRemove < 0) {
71
72
                  data[blockNo][componentRef].karma = karmaToAdd;
73
              } else {
74
                   data[blockNo][componentRef].karma = currentKarma - karmaToRemove + karmaToAdd;
```

```
}
 75
76
77
78
                 meta[blockNo].totalKarma += data[blockNo][componentRef].karma;
 79
                 // The karma generated is divided evenly over its contributors
                 int64 karmaPerCollaborator = data[blockNo][componentRef].karma / int64(numCollaborators);
 80
                 for (uint i = 0; i < numCollaborators; i++) {</pre>
 81
                      bytes32 collaborator = componentDir.getComponentCollaborator(componentRef, i);
data[blockNo][collaborator].karma += karmaPerCollaborator;
 ^{82}
 83
                      data[blockNo][collaborator].karmaParts.push(KarmaPart(componentRef, karmaPerCollaborator));
 84
 85
 86
                      delete data[blockNo-windowSize][collaborator];
                 }
 87
 88
                 delete data[blockNo-windowSize][componentRef];
 89
            }
 90
 ^{91}
 ^{92}
            function mineWavelets(uint32 blockNo, bytes32[] refs, bool last) {
 93
                 if (meta[blockNo].state != BlockState.Sealed) { return; }
                 for (uint i = 0; i < refs.length; i++) {
    address addr = userDir.getWalletAddress(refs[i]);
    if (addr == address(0x0) || data[blockNo][refs[i]].karma == 0 || data[blockNo][refs[i]].mined) {</pre>
 ^{94}
 95
 96
 97
                           continue;
                      }
 ^{98}
99
100
                      uint share = blockReward * uint(data[blockNo][refs[i]].karma) / uint(meta[blockNo].totalKarma);
101
                      wavelets.mint(addr, share, blockNo);
data[blockNo][refs[i]].mined = true;
102
103
104
                 }
105
                 if (last) {
106
                      meta[blockNo].state = BlockState.Mined;
lastBlock = blockNo;
107
108
                 }
109
            }
110
111
      }
```

```
contract Token {
 1
          uint256 public totalSupply;
2
          function balanceOf(address _owner) constant returns (uint256 balance);
3
          function transfer(address _to, uint256 _value) returns (bool success);
4
          function transferFrom(address _from, address _to, uint256 _value) returns (bool success);
5
          function approve(address _spender, uint256 _value) returns (bool success);
 6
          function allowance(address _owner, address _spender) constant returns (uint256 remaining);
7
          event Transfer(address indexed _from, address indexed _to, uint256 _value);
event Approval(address indexed _owner, address indexed _spender, uint256 _value);
 8
9
10
     }
11
      contract StandardToken is Token {
12
^{13}
          function transfer(address _to, uint256 _value) returns (bool success) {
               if (balances[msg.sender] >= _value && _value > 0) {
    balances[msg.sender] -= _value;
14
15
                    balances[_to] += _value;
16
                    Transfer(msg.sender, _to, _value);
17
18
                   return true;
19
               } else {
20
                   return false;
               }
21
          }
^{22}
23
          function transferFrom(address _from, address _to, uint256 _value) returns (bool success) {
^{24}
^{25}
               if (balances[_from] >= _value && allowed[_from][msg.sender] >= _value && _value > 0) {
                   balances[_to] += _value;
balances[_from] -= _value;
26
27
                    allowed[_from][msg.sender] -= _value;
28
                    Transfer(_from, _to, _value);
^{29}
30
                    return true;
31
               } else {
32
                   return false;
33
               }
          }
34
35
          function balanceOf(address _owner) constant returns (uint256 balance) {
36
37
               return balances[_owner];
38
          3
39
          function approve(address _spender, uint256 _value) returns (bool success) {
    allowed[msg.sender][_spender] = _value;
    Approval(msg.sender, _spender, _value);
    return true;
40
41
42
^{43}
               return true;
^{44}
          7
45
          function allowance(address _owner, address _spender) constant returns (uint256 remaining) {
46
               return allowed[_owner][_spender];
47
^{48}
^{49}
50
          mapping (address => uint256) balances;
          mapping (address => mapping (address => uint256)) allowed;
51
     }
52
53
      contract Wavelets is StandardToken {
54
          string public constant name = "Wavelets";
55
          string public constant symbol = "WVT";
56
          uint256 public constant decimals = 18;
string public version = "1.0";
57
58
          ACL acl;
59
          event Mint(address indexed _to, uint256 _value, uint32 _blockNo);
60
61
          function Wavelets(ACL _acl) {
62
63
               acl = _acl;
64
          3
65
          function mint(address recipient, uint256 value, uint32 blockNo) {
66
               if (!acl.hasRole(tx.origin, "admin")) { return; }
67
               totalSupply += value;
68
               balances[recipient] += value;
69
70
               Mint(recipient, value, blockNo);
71
          }
     }
72
```

```
contract ComponentDirectory {
 1
             struct Component {
 2
                  uint8 _type;
string id;
 3
 4
 \mathbf{5}
                   bytes32[] collaborators;
 6
             }
 \overline{7}
             ACL acl;
 8
9
             mapping(bytes32 => Component) public components;
             bytes32[] componentRefs;
10
11
             uint componentCount;
^{12}
             function ComponentDirectory(ACL _acl) {
13
^{14}
                   acl = _acl;
             }
15
16
17
             function registerComponent(bytes32 ref, uint8 _type, string id, bytes32[] collaborators) {
18
                   if (!acl.hasRole(tx.origin, "admin")) { return; }
19
                   if (components[ref]._type > 0) { return; }
^{20}
21
                   components[ref] = Component(_type, id, collaborators);
componentRefs.push(ref);
^{22}
^{23}
^{24}
                   componentCount++;
             }
^{25}
^{26}
             function unregisterComponent(bytes32 ref) {
    if (!acl.hasRole(tx.origin, <sup>2</sup>admin<sup>2</sup>)) { return; }
\frac{27}{28}
                   delete components[ref];
^{29}
30
                   for (uint i = 0; i < componentCount; i++) {
    if (componentRefs[i] == ref) {
        componentRefs[i] = componentRefs[componentRefs.length-1];</pre>
^{31}
^{32}
33
                              componentCount--;
34
35
                              return;
36
                         }
37
                   }
             }
38
39
             function setCollaborators(bytes32 ref, bytes32[] collaborators) {
    if (!acl.hasRole(tx.origin, [']admin'])) { return; }
    components[ref].collaborators = collaborators;
40
41
^{42}
^{43}
             }
       }
^{44}
```

```
1
      contract UserDirectory {
          struct User {
2
               string name;
3
               address walletAddress;
4
\mathbf{5}
          }
 6
 \overline{7}
          ACL acl;
          mapping(bytes32 => User) public users;
mapping(address => bytes32) public addresses;
 8
9
          bytes32[] userRefs;
10
11
          uint userCount;
^{12}
          function UserDirectory(ACL _acl) {
13
^{14}
               acl = _acl;
          }
15
16
17
          function registerUser(bytes32 ref, string name, address walletAddress) {
18
               if (!acl.hasRole(tx.origin, "admin")) { return; }
19
               if (bytes(users[ref].name).length > 0) {
^{20}
^{21}
                    return;
               }
22
^{23}
^{24}
               userRefs.push(ref);
^{25}
               users[ref].name = name;
^{26}
               if (walletAddress != address(0x0)) {
                    users[ref].walletAddress = walletAddress;
^{27}
               }
^{28}
^{29}
               userCount++;
30
          }
^{31}
          function unregisterUser(bytes32 ref) {
^{32}
               if (!acl.hasRole(tx.origin, "admin")) { return; }
33
34
               if (bytes(users[ref].name).length == 0) {
35
36
                    return;
               }
37
38
               delete users[ref];
39
40
               for (uint i = 0; i < userCount; i++) {
    if (userRefs[i] != ref) {</pre>
41
^{42}
^{43}
                         continue;
                    3
^{44}
^{45}
                    userRefs[i] = userRefs[userCount-1];
46
47
                    userCount--;
^{48}
^{49}
                    return;
50
               }
          }
51
52
          function setWalletAddress(bytes32 ref, address walletAddress) {
53
               if (!acl.hasRole(tx.orgin, "admin")) { return; }
users[ref].walletAddress = walletAddress;
54
55
               addresses[walletAddress] = ref;
56
57
          }
     }
58
```

Appendix B

Performance measurements

Run	Query (s)	Submit (s)	Finalize (s)	Total (s)
1	1.09	1118.53	68.97	1188.59
2	1.06	785.74	54.04	840.84
3	1.07	769.12	47.69	817.88
4	0.67	1423.98	72.23	1496.88
5	1.25	1584.80	75.69	1661.74
6	0.54	1244.05	54.84	1299.43
7	0.60	1008.48	57.92	1067.00
8	0.58	1123.82	58.92	1183.32
9	0.53	956.95	52.29	1009.77
10	0.58	1199.23	74.14	1273.95
11	0.96	1632.09	71.10	1704.15
12	0.67	1300.21	93.22	1394.10
13	0.47	1149.37	80.01	1229.85
14	0.53	1598.02	66.56	1665.11
15	0.56	989.79	51.97	1042.32
16	0.51	976.80	80.80	1058.11
17	0.39	1583.02	58.59	1642.00
18	0.53	880.52	54.35	935.40
19	0.10	769.72	78.58	848.40
20	0.86	1712.16	79.40	1792.42
21	0.40	1230.27	51.00	1281.67
22	0.53	621.34	45.76	667.63
23	0.50	993.13	57.82	1051.45
24	0.54	1147.60	52.10	1200.24
Mean	0.65	1158.28	64.08	1223.01
SD	0.26	299.52	12.63	305.47
Ratio	0.05%	94.71%	5.24%	100%

Table B.1: Durations of various stages of the batch process

A Blockchain-Based Micro Economy Platform for Distributed Infrastructure Initiatives

Jan Kramer[†], Jan Martijn E. M. van der Werf^{*} and TBD

*Utrecht University, Princetonplein 5, 3584 CC Utrecht, Netherlands Email: {j.m.e.m.vanderwerf, s.brinkkemper}@uu.nl †The Things Network, Herengracht 182, 1016 BR Amsterdam, Netherlands Email: jan@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 validated through a case study at TTN, where a proof of concept of the architecture was implemented and evaluated. The case study revealed that the architecture is effective for the given situation, but needs more research in the areas of scalability and security to be deployed on a larger scale.

Index Terms—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-topeer 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, given basic economic principles, voluntary contributions to such initiatives are limited and potentially unsustainable.

Developing a micro economy specific to DIIs can address this issue by enabling incentives to be set up so that contributions are rewarded. However, while there is a vast body of literature on related topics such as reward systems, there is little practical tooling to enable DIIs develop such micro economies.

Therefore, in this research we aim to provide a software architecture of a platform that provides the tools necessary to operationalize an economic model that incentivizes participants to contribute beyond their own needs and altruism. To achieve a fully decentralized architecture, the platform is centered around a blockchain, a type of distributed ledger, using smart contracts to facilitate the core logic. Finally, the architecture is validated by implementing a proof of concept and evaluating it in the context of TTN, a real-world DII.

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. 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 [4]. 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-A.

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 [5], or to the actual instantiation of the pattern as applied in projects such as Bitcoin and Ethereum [6]. For the purpose of this research, we are mainly interested in the generic architectural pattern of a distributed ledger.

A. 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 [5]. 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 [7]. Alternative approaches that do not incur a severe pressure on the environment include Proof-of-Space [8]–[10], which is based on miners providing disk space instead of computational resources, Proof-of-Stake [11], 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.

B. Smart Contracts

Another aspect that varies across distributed ledgers is whether they support *smart contracts*. While popularized by Ethereum [12], Smart Contracts were actually first defined in [13]. 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 [6], [12]. 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.

C. 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 [14]–[16] 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 [16]. 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 contributor can simultaneously be 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.

- 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 DIIspecific 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-C2. 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 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 [17].

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-C2. Finally, a third group of outside actors, investors, might exchange tokens with participants without actively participating in the network.



Fig. 1. Context diagram of micro economy platform

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 an hourly basis, 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.



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

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-C2. 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

Figure 3 depicts the high-level platform design 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 temporarily stores the metrics. 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, because 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*. 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*. The services are user defined, but an example of a possible service is provided in Section III-C2.



Fig. 3. High-level platform structure

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.

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 them, although the platform could provide template contracts.



Fig. 4. Token reward workflow

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 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.



Fig. 5. User defined SLA workflow

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 selfdestruct the service. Any remaining tokens in escrow are returned to the users.

3) 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 discussed in Section III-C 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 we can fully trust. 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 architecture, we conducted a case study in the context of 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 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



Fig. 7. Typical usage scenario of The Things Network

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.

Figure 8 provides an overview of which aspects have been implemented and which have been omitted. 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.



Fig. 8. Overview of components. Implemented components are highlighted (e.g. Monitor)

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. 9. Monitor implementation

Monitor Figure 9 depicts a more detailed view of the implemented monitor. Given that in the context of TTN infrastructure components already expose data over gRPC¹ streams and bindings for these streams are available in the Go programming language², 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 InfluxDB³, 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.

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

¹See https://grpc.io/about/

²See https://golang.org/

As depicted in Figure 10, 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 the username-address combination on the blockchain in the Directory contract, after which the identification procedure is finished.



Fig. 10. 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.

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.

Implemented Smart Contracts



Fig. 11. Overview of smart contracts and their relationships

Figure 11 depicts the actually implemented smart contracts as a simplified UML class diagram.

Although the architecture suggest a separation between the Karma and Infrastructure Metrics Store contracts (cf Figure 8), 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-C1, 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. To that end, 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, but 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 24hour 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.

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 [18] 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 [19], 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 [20]. 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.

2) Security: Given that rewards contributors receive constitute some 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 wellmaintained. Vulnerabilities might occur in these components, but can largely be mitigated by keeping the software up-todate.

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 still 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 wellknown 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 that still does not provide any watertight proof on the 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 that based on our initial findings appears to be effective. However, the implementation has revealed several areas that should be researched more extensively.

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 [20]) to reduce the storage requirements, and 2) explore the possibility of using *payment channels* [21] for smart contracts to reduce the number of required transactions and thereby increasing the overall throughput.

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

- J. Hamari, M. Sjöklint, and A. Ukkonen, "The sharing economy: Why people participate in collaborative consumption," *Journal of the Association for Information Science and Technology*, vol. 67, no. 9, pp. 2047–2059, 2016.
- [2] A. Sundararajan, "The power of connection: Peer-to-peer businesses," 2014. [Online]. Available: https://smallbusiness.house.gov/UploadedFiles/1-15-2014_Revised_Sundararajan_Testimony.pdf
- [3] The things network. [Online]. Available: https://thethingsnetwork.org
- [4] M. Swan, Blockchain: Blueprint for a new economy. "O'Reilly Media, Inc.", 2015.
- [5] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf
- [6] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," 2014.
- [7] 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.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 135–156.
- [8] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak, *Proofs of Space*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 585–605.
- [9] 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. [Online]. Available: https://www.microsoft.com/en-us/research/publication/permacoin-repurposing-bitcoin-work-for-data-preservation/
- [10] 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.
- [11] I. Bentov, A. Gabizon, and A. Mizrahi, *Cryptocurrencies Without Proof* of Work. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 142–157.
- [12] V. Buterin, "A next-generation smart contract and decentralized application platform," 2014. [Online]. Available: https://www.ethereum.org/pdfs/EthereumWhitePaper.pdf
- [13] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, 1997.
- S. D. Lerner, "Dagcoin: a cryptocurrency without blocks," 2015. [Online]. Available: https://bitslog.files.wordpress.com/2015/09/dagcoinv41.pdf
- [15] A. Churyumov, "Byteball: a decentralized system for transfer of value," 2015. [Online]. Available: https://byteball.org/Byteball.pdf
- [16] S. Popov, "The tangle," 2016. [Online]. Available: https://iotatoken.com/IOTA_Whitepaper.pdf
- [17] N. Rozanski and E. Woods, Software systems architecture: working with stakeholders using viewpoints and perspectives, 2nd ed. Addison-Wesley, 2012.
- [18] "State tree pruning." [Online]. Available: https://blog.ethereum.org/2015/06/26/state-tree-pruning/
- [19] D. Frey, M. X. Makkes, P.-L. Roman, F. Taïani, and S. Voulgaris, "Bringing secure bitcoin transactions to your smartphone," in *Proceedings of the 15th International Workshop* on Adaptive and Reflective Middleware, ser. ARM 2016. New York, NY, USA: ACM, 2016, pp. 3:1–3:6. [Online]. Available: http://doi.acm.org/10.1145/3008167.3008170
- [20] J. Benet, "IPFS content addressed, versioned, P2P file system," *CoRR*, vol. abs/1407.3561, 2014. [Online]. Available: http://arxiv.org/abs/1407.3561
- [21] C. Decker and R. Wattenhofer, A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels. Cham: Springer International Publishing, 2015, pp. 3–18.