Use of IOTA 1.0 for Micropayments in Transport and IoT

Pedro Miguel Fernandes de Bastos Sécio Gomes

Thesis to obtain the Master of Science Degree in

Information Systems and Software Engineering

Supervisor(s): Prof. José Manuel da Costa Alves Marques
Dr. Joel dos Santos Teixeira

Examination Committee

Supervisor: Prof. João António Madeiras Pereira
Chairperson: Prof. José Manuel da Costa Alves Marques
Member of the Committee: Prof. Paolo Romano

September 2020

Dedicated to my mother.

Acknowledgments

I would like to express my sincere gratitude to everyone who helped me in this process. In no particular order, I would like to thank:

My advisor, Professor José Manuel da Costa Alves Marques, for the guidance throughout this work and for all the support and availability shown giving me relevant and essential contributions to the development, with his knowledge and high competence, of this research.

My co-advisor, Joel dos Santos Teixeira, for the way he has always received and supported me, leading me through his advice and knowledge to the end of this work.

The Link Consulting Employees for their demonstrated availability, relevant suggestions, and support provided in the course of this work, namely Hugo Miguel Bicho.

The the Management of Link Consulting for having opened the way for the investigation of this topic that is so pertinent in view of the new demands of the current world.

My family for fully supporting and providing me with the opportunity to be here. And above all, because everything I am and accomplish would not be possible without them, especially, my parents and brother for the constant motivation throughout life, love and dedication.

Finally, my friends and colleagues, for the company and exchange of knowledge throughout all these years of course and life. Thank you all very much!

Resumo

Menos de 12 anos após o seu início, as moedas criptográficas emergiram da obscuridade para empolgar empresas e consumidores, bem como os bancos centrais e outras autoridades. Estas empresas atraem porque prometem substituir a confiança em instituições de longa data, pela confiança num sistema novo, totalmente descentralizado, baseado na blockchain e na tecnologia de ledgers distribuídos (DLT). Esta tecnologia apresenta custos de processamento consideráveis, suportáveis para transacções financeiras de elevado valor, mas desadequada para apoiar numerosas transacções de baixo valor. Foram propostas novas variantes dos algoritmos utilizados inicialmente para permitir a implementação de micropagamentos nas transacções Machine-to-Machine (M2M) associadas à Internet das Coisas (IoT), como no caso da IOTA, uma moeda digital que promete ser adequada para suportar pagamentos associados à IoT, o que evita o peso associado à implementação da blockchain, mantendo a validação distribuída das transacções. Assim, pretendeu-se avaliar a adequação do IOTA para a implementação de micropagamentos na área do IoT aplicado aos transportes públicos. Como base, tentámos tirar partido da imutabilidade das fichas emitidas, permitindo o registo das transacções de forma assíncrona, mantendo a validação e contabilização destas transacções de uma forma segura e distribuída. Na solução proposta os vários modelos de implementação do IOTA na sua aplicação aos transportes públicos foram estudados e aprofundados pelas suas limitações e vantagens. Os resultados do trabalho demonstraram uma solução com capacidade de processamento adequada apresentando, no entanto, bastantes limitações para implementação num caso real, no seu estado actual.

Palavras-chave: Blockchain, Bilhete integrado, Internet das Coisas, IOTA, Micropagamentos, Tecnologia de Ledger Distribuído

Abstract

Less than 12 years after their inception, cryptographic coins have emerged from obscurity to excite businesses and consumers, as well as central banks and other authorities. They attract because they promise to replace trust in long-standing institutions with trust in a new, fully decentralised system based on blockchain and distributed ledger technology (DLTs). This technology has considerable processing costs, bearable for high-value financial transactions, but unsuitable to support numerous low-value transactions. New variants of the algorithms initially used to allow the implementation of micro-payments in Machine-to-Machine (M2M) transactions associated with the Internet of Things (IoT) have been proposed, as in the case of IOTA, a digital currency that promises to be suitable to support payments associated with IoT, which avoids the weight associated with implementing the blockchain while maintaining distributed validation of transactions. The aim was to assess the suitability of the IOTA for the implementation of micro-payments in the area of IoT applied to public transport. As a basis, we intended to take advantage of the immutability of the issued tokens, allowing the registration of transactions in an asynchronous way, keeping the validation and accounting of these transactions in a safe and distributed manner. In the proposed solution, the various models for implementing IOTA in its application to public transport have been studied and deepened due to their limitations and advantages. The results of the work have demonstrated a solution with adequate processing capacity, but with many limitations for implementation in a real case scenario, in its current state.

Keywords: Blockchain, Distributed Ledger Technology, Integrated ticketing, Internet of Things, IOTA, Micropayments

Contents

List of Tables

List of Figures

Nomenclature

CPU	Central Processing Unit.
DLT	Distributed Ledger Technology.
DNS	Domain Name System.
HTTP	Hyper Text Transfer Protocol.
HTTPS	Hyper Text Transfer Protocol Secure.
IO	Input/Output.
IoT	Internet of Things.
IRI	IOTA reference implementation.
LAN	Local Area Network.
M2M	Machine to Machine.
RAM	Random-access Memory.
REST	Representational State Transfer.
SDK	Software Development Kit
TCP	Transmission Control Protocol.

Chapter 1

Introduction

Blockchain, IOTA and other forms of Distributed Ledger Technologies (DLT) combine recent advances in data science, cryptography, and novel governance principles - and have been highlighted as one of the most disruptive sets of technologies since the advent of the internet. The DLT is based on a shared, encrypted database to store, protect and validate electronic transactions, without the need for a central validation system. In this way, it presents itself as a decentralised, open and public ”ledger system”, similar to a database, whose validation is done, according to a specific consensus protocol, by its users. The great leap in popularity of the DLT took place in 2007 with the creation of Bitcoin Blockchain. However, when looking at its applicability to the Internet of Things (IoT), issues such as scalability, offline accessibility, transaction fees and quantum security were not resolved. Alternatively, the IOTA Foundation developed and published the tangle, which it claimed resolved these issues inherent in the Blockchain. It then intended, with the development of this Tangle technology, to equip IOTA with an appropriate distributed ledger. A scalable ledger was essential to be able to handle the vast number of transactions sent by millions of devices, including micro-transactions and zero value transactions, as messages or sensor data. In other words, if they had to pay fees for every piece of data a device sent to their peers, these transactions would become very expensive for numerous IoT scenarios. For many, it then became apparent that Tangle, while presenting very similar characteristics to Blockchain, would allegedly be able to solve all the problems mentioned above. This growing popularity of DLT in recent years has triggered a wave of innovations, experiments, analysis and research, which has sparked, among others, the financial sector, which has been faced with the emergence of a vast number of explorations made using this technology for payments and settlements. These experiences with DLT demonstrate their potential for the next generation of payment systems, improving the integration and reconciliation of settlement accounts and their ledgers, involving electronic and mobile payment programmes that enable integrated payments, in real-time, flatter structures, continuous operations and global reach, and end-to-end payment and settlement transfers. Although this technology is mainly used for monetary transactions, for the time being, experts have already begun to perceive and study its usefulness in changing the mobility sector. European Commissioner for Transport, Violeta Bulc. recognised that it could help in the challenge of integrating shared mobility services. Applications in this area allow actors to establish direct relations between themselves according to a commonly agreed set of rules and a high degree of trust, without having to go through a central authority. By combining a common language and syntax for the ”mobility internet” and new means of accounting transactions, these applications can help redefine the way people and organisations access, pay for, use and/or manage mobility services through a wide network of unrelated and competing transport service providers and platforms. Much in Mobility as the Service ecosystem is ”blockchain-able”.

However, Mobility as a Service goes far beyond getting a person from A-to-B; it is about understanding the journey and getting a person from A-to-Z. Moblity as a Service (MaaS) is possible through three core technologies: mobile, massive data, and the Internet of Things (IoT). More and more people interact with the technological world through the main screen: their smartphone. A mobile device contains the front-end of the MaaS experience. Still, this high dependency on smartphones would not be possible without greater connectivity thanks to advances in wireless communications technology. The ubiquity of this wireless network gives rise to the Internet of Things, a wireless network of physical devices, vehicles and devices with which people interact in real-time, integrated into the fabric of our world. Although not part of this project, sensors connected to physical objects in the world of mobility influence the way transport systems, personal vehicles, traffic control and even intelligent car parks work. In the future, DLT could support this exponentially increasing volume of data and machine learning, and AI would become necessary to obtain intelligent knowledge that could increase the value of services by providing a seamless MaaS ecosystem.

This research and experience on the use of DLT for micropayments in a transportation ticketing system has provided information on its potential benefits, risks, limitations, and implementation challenges, considering that, despite its limitations and lack of maturity, this technology presents great potential for implementation and likely long-term applications and benefits for the development of the micropayment system in IoT, in future versions.

1.1 Objectives

The main goal of this thesis is to study the applicability of IOTA in the mobility sector by conducting a series of benchmarks on the technology and developing a Proof of Concept to prove the feasibility of the designed solution. It aspires to meet the following objectives:

- Understand how well IOTA integrates with existing different operation models
- Develop a seamless proof-of-concept demonstrator for the practical validation of results
- Explore the resources needed for its practical implementation

1.2 Thesis Outline

This dissertation is structured according to the research issues mentioned above. The first chapter after the introduction covers a literature review on both IoT distributed ledger technology applications as their applications in urban mobility, and the fundamentals of the concept ”Mobility as a Service”. It will also focus on how ticketing and payment transactions, and processing and clearing mechanisms through distributed ledger technologies can change. Finally, a description of different DLTs and a comparison in terms of suitability for IoT (and indirectly, MaaS). Section 3 will detail the operation of the chosen technology, IOTA, to be addressed in this dissertation. Section 4 will present the implemented solution and proposed architecture decisions for an IOTA-based ticketing system. Section 5 then details the implementation details for the previously presented solution. Section 6 demonstrates test results for the IOTA technology and the proposed solution and will explain how to assess its benefits and limitations. Finally, section 7 expresses some personal conclusions on this project.

Chapter 2

Background and Related Work

2.1 Internet of Things

The continuous search for solutions that result in increased well-being and quality of life for the population, the sustainability of organisations, families, and the environment, has led, among other things, to the search for and implementation of solutions based on connecting devices to the Internet.

The Internet of Things (IoT) emerges, then, as a ”system that interconnects computing devices, be they computers, machines, people, animals, or objects, and within which they can communicate and transfer data without any human intervention” [1]. In other words, it is a system formed by networks, containing countless quantities of connected devices and, since most of these devices interact through a central entity, IoT presents a centralised structure. IoT enables objectives to be monitored and triggered/concreted remotely from the network, allowing full integration between computer systems and the physical world. In recent years, we have witnessed a revolution of IoT, with the connection of all kinds of objects in the network and the exponential growth of the number of devices connected.

However, this tremendous growth in a combined centralised system raises new challenges such as security and privacy [2], scalability and data processing performance for IoT System Architecture, which means that an effective solution has to be designed, namely a direct peer-2-peer (P2P) interaction, without always using a central entity, can be important for the continued growth of IoT. [3]

2.2 Micropayments

Micropayments are a type of device-device interaction, defined as a ”payment of a small amount” [4], e.g., a fraction of a cent . They are often issued as successive partial payments while using a service or product (i.e., pay-per-use). Micropayments are an alternative to traditional charging models as they allow these payments to be made more dynamically using mobile media or native internet environments. ”However, issuing autonomous P2P micropayments has its challenges, one of them being that it would require a technology that is both secure and scalable” [5]. Therefore, since there is still no standardised payment solution for making micropayments between Internet of Things (IoT) devices, it is essential to

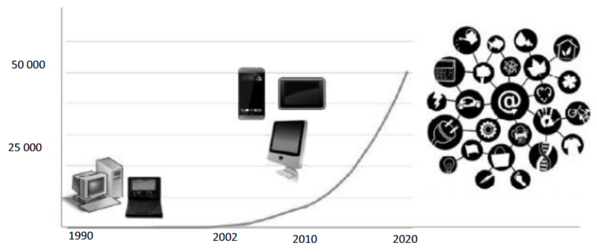


Figure 2.1: Evolution of devices connected to the Internet in [1]

continue the search for a sustainable solution, driving the emergence of pay-per-use business models.

2.3 Distributed Ledger Technology

Distributed Ledger Technology (DLT) is defined by Prableen Bajpai [6] as ”... the technological infrastructure and protocol that allow simultaneous access, validation and record updating in an immutable manner across a network spread across multiple entities or locations”. The DLT has attracted the attention of industry and academia, as its characteristics of decentralisation, immutability, and scalability have given it the potential to manage and register secure micropayments, constituting a possibility to overcome problems in IOT systems [3]. According to different data structures for the ledger, there are fundamentally two main types of DLT, blockchain based on blocks or like a directed acyclic graph (DAG) where there are no blocks (for example, IOTA Tangle [7]).

Blockchain is a distributed ledger for storing and sharing data across all nodes in a network [3]. This ledger is considered tamper-proof, making it exceptionally suitable for its main area of application - cryptocurrencies, for example, Bitcoin, which, however, has posed problems of scalability, as it is slow and expensive to perform transactions in the Bitcoin blockchain, due to the single chain of blocks being linear, and the blocks cannot be created simultaneously. These limitations have led to the development of specialised adaptations and changes to the original Bitcoin blockchain, as well as the development of alternative DLTs [5].

2.4 Distributed Ledger Technology in Cryptocurrencies

Cryptocurrencies decentralised record-keeping system is known as a distributed ledger. An up-to-date copy of the entire ledger is stored by each participant or node in the network. With a distributed ledger, peer-to-peer exchange of digital money is feasible: each participant can directly verify in their accessible copy of the ledger whether a transfer took place or not and if there was no attempt to double-spend (Any digital form of money is easily replicable and can thus be fraudulently spent more than once) [8]. The blockchain is a distributed ledger that is updated in groups of transactions called blocks. Blocks are then chained sequentially via the use of cryptographic hashes to form the blockchain. This concept has been adapted to numerous other cryptocurrencies. Nonetheless, these share many core features. The case of blockchain helps illustrate how these distributed ledgers function. Blockchains have seven principal

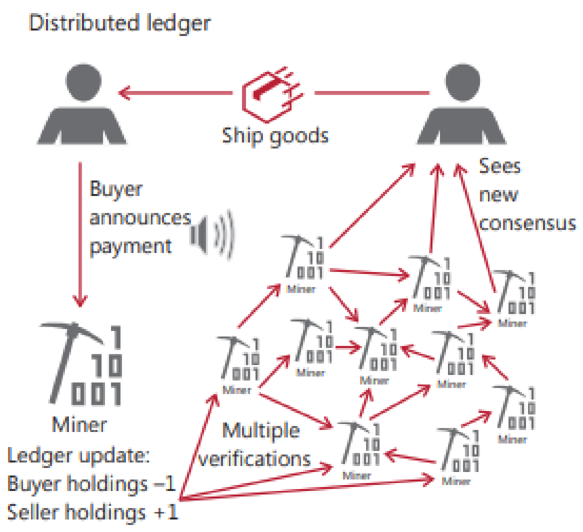


Figure 2.2: Valid transactions in a permissionless cryptocurrency [8]

characteristics:

- Distributed databases and ledgers;
- Irreversibility of records;
- Transparent identity management with pseudonymity;
- Robust validation and consensus;
- Peer-to-peer transmission;
- Computational logic.

Blockchain-based permissionless cryptocurrencies have two groups of participants: “miners” who act as bookkeepers and “users” who want to transact in the cryptocurrency. At face value, the idea underlying these cryptocurrencies is simple: the ledger is updated by a miner, and the update is subsequently stored by all users and miners. [9]

Underlying this setup, the key feature of these cryptocurrencies is the implementation of a set of rules (the protocol) that aim to align the incentives of all participants to create a reliable decentralised payment technology. The protocol determines the supply of the asset to counter debasement – for example, in the case of Bitcoin, it states that no more than 21 million bitcoins can exist. In addition, the protocol is designed to ensure that all participants follow the rules out of self-interest, i.e. that they yield a self-sustaining equilibrium. Three key aspects are the following. [8]

First, the rules entail a cost to updating the ledger. In most cases, this cost comes about because updating requires a “proof-of-work”. This is mathematical evidence that a certain amount of computational work has been done, in turn calling for costly equipment and electricity use often referred to as mining. In return for their efforts, miners receive fees from the users – and, if specified by the protocol, newly minted cryptocurrency.

Second, all miners and users of a cryptocurrency verify all ledger updates, which induces miners to include only valid transactions. Valid transactions need to be initiated by the owners of funds and must not attempt to double-spend. If a ledger update includes an invalid transaction, it is rejected by the network, and the miner's rewards are voided. The verification of all new ledger updates by the network of miners and users is thus essential to incentivise miners to add only valid transactions.

Third, the protocol specifies rules to achieve a consensus on the order of updates to the ledger. This is generally done by creating incentives for individual miners to follow the computing majority of all other miners when they implement updates. Such coordination is needed, for example, to resolve cases where communication lags lead to different miners adding conflicting updates – i.e. updates that include different sets of transactions. The way DLTs are set up replaces trust between different parties or trust in some form of an oversight committee with cryptographic proof of validity or "consensus". It proposes that any transaction could be authenticated, and any transmitted piece of information maintained by an emergent process of consensus among a globally distributed network of peers that follow a precise, incorruptible method to check any change in the system. The cryptographic identity of each new block in a blockchain must be validated (recognized as authentic, in conformity with the cumulative blockchain and linked to a unique identity) before it can be included in the latest iteration of the ledger that is propagated to, and recognized by, all nodes. These "consensus" protocols and algorithms are linked to the nature of the blockchain in question.

Finally, once a transaction has entered in the database and the accounts are updated, the records cannot be altered, because they're linked to every transaction record that came before them (hence the term "chain"). The recording of the blockchain database at any given time is permanent, chronologically ordered, and available to all others on the network. This immutability is at the heart of the "trustfulness" of the blockchain.

What makes this technology so appealing and game changing is the absence of third parties [10], such as payment processors, during the exchanges. This means that for a transaction to be made, it has to be validated by the community (peer-to-peer).

2.5 Decentralized applications

Decentralised applications running in peer-to-peer networks built on distributed ledger, blockchain and other novel data protocols are starting to profoundly disrupt established economic sectors (e.g. finance – Securrency, Circle; healthcare – MedicalChain, MedRec; insurance – Accenture; commerce – Origin-Trail, De Beers) [11]. These applications allow agents to enter into direct relationships with each other according to a commonly agreed set of rules and a high degree of trust. These applications, combined with a common language and syntax for the "Internet of mobility" and new means of deriving insight from previously siloed data, may help redefine how people access, pay for and use transport in their everyday lives.

Today, choices available to citizens are expanding and changing rapidly in an ever more dynamic urban mobility ecosystem. Traditional stakeholders are facing increasing pressure to innovate, to attract

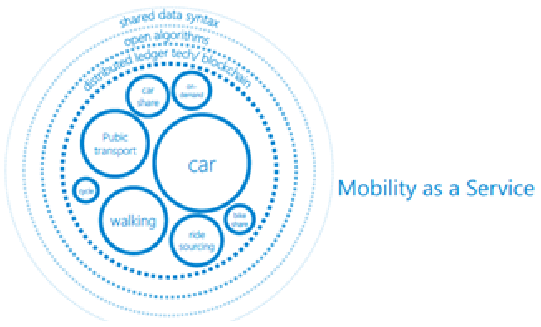


Figure 2.3: Mobility as a Service Leverages digitalization for customer-centric transport services [11]

and retain users and, to do so, must alternatively compete and co-operate with each other and new market entrants proposing novel business models [11]. Against this backdrop is the consumer who just wants to get from point A to point B in the most demand responsive, flexible, pain-free, reliable and affordable way. As in other areas of their lives, they want trip experiences that place them in control, and which leverage the most convenient options available irrespective of who offers them.

Car-use has generally responded well to peoples desire for seamless, convenient and comfortable travel across a broad range of distances and in many urban contexts. Further, the affordability of car use has grown alongside growth in incomes and lower relative travel costs. But this growth has come at a cost to cities, people and society (e.g. congestion, unreliability of travel time, air pollution, crashes, car-dependency, inequitable access) that have eroded the benefits cars provide.

This growing tension between cars and mobility in cities has led cities, citizens and companies to explore ways in which the benefits of car-like mobility can better be delivered across a wide range of mobility options by leveraging digital assets and data science. At the heart of these efforts are the many ways in which private and public actors are seeking to offer a seamless and rich Mobility as a Service (MaaS) offer that could compete effectively with – or integrate – private car-based mobility [11].

2.6 Mobility as a Service

2.6.1 Overview

Mobility as a Service (MaaS) [12] is a term used interchangeably to describe packages of bundled transport services or, more generally, as a broad concept describing new, customer-centric ways of seamlessly accessing a range of different transport services – many of them shared. MaaS represents a break with the past in that mobility services have historically been provided by siloed operators, manufacturers and public authorities with little practical cross-mode coordination.

At its core, the concept of MaaS supports the digital joining-up of different transport, information and payment services into a smooth and reliable customer-facing experience (Figure 2). These services may be those provided by a single operator in cases where extensive integration exists or may involve a MaaS provider bringing together services offered by third parties into a coherent framework [11].

MaaS requires a set of transparent, vetted and trusted commercial agreements that encompass commercial operators, public services and third-party aggregators of services (where applicable) and should cover payment and revenue allocation amongst all parties. These agreements should enable viable services to be developed by all parties.

2.6.2 Everything-to-everything Interoperability

Delivering on the promise of MaaS in a meshed world of customers, transport operators, data providers, infrastructure and asset owners, and public authorities will require a significant shift away from the status quo covering existing protocols and business logic. Ensuring this shift enables “everything-to-everything” interoperability in the context of MaaS will involve action in the following three areas [11]:

- Distributed Ledger Technology (DLT): A move away from platform based MaaS frameworks. Central to these frameworks is the way in which distributed and non-centralised trust, robust identification, authentication functions and payment transactions and clearing are carried out. Using DLTs, urban mobility could then be made possible without the need for an intermediary (e.g, clearing house). In this scenario, clients would pay to have digital currency/tokens, and every utilization of the transport would be handled as a transaction to a specific service provider, which had no need to be validated by a third party since DLTs already provide the ensurance that the transaction was correct and real. Such a ticketing and transactional mechanism would help reduce fees, while also providing a concrete seamless solution. Therefore, DTLs definitely are MaaS enablers;
- Data syntax for MaaS. A common data syntax for encoding the various components of MaaS would facilitate the uptake of these services. Such a shared data syntax would be used as a basis for encoding transport services in a blockchain/distributed ledger environment and would thus represent the building blocks for seamless MaaS;
- Open Algorithms and other alternatives to data-sharing. Alternatives to data sharing that enable stakeholders to access vetted, trusted and actionable insight from proprietary data may remove many roadblocks to broad MaaS partnerships.

2.6.3 Revenue allocation

In a Mobility as a Service environment, new revenue allocation mechanisms must address how service providers/operators are to be compensated for their fractional contribution to a total trip chain. These mechanisms should allow all parties to achieve consensus on what resources were used to fulfil a trip and how payments for these were allocated across all actors.

Pay as you go

The Pay-as-you-go format becomes then possible, overcoming the shortcomings of current payment methods enabling users to spend for their commute on-demand and when they need too, without any worries of prepaid balance limitations or losing the balance in case of expiry. The tariff should not be linked to a mode of transport, neither to a technology, but to the concept of an Origin-Destination displacement from the perspective of the passenger, involving multiple transports. The accounting on the other hand, is calculated by the intensity of use and not just by the simple sum of the uses.

2.6.4 Benefits

In this section, the retailers are considered the entity that provides Mobility as a Service. As such, we present some of its benefits:

- The retailer cannot be held accountable for creating tickets incorrectly because the data syntax is the same between operators (i.e., all operators share the same data syntax, therefore if a ticket does not follow its syntax, it is the operator's fault);
- The barrier for entry to market is much lower as operators do not need to understand the details of every ticket format and there is much less need for an accreditation process and financial bonds (i.e., the trust is placed on the system designed to enable MaaS);
- Providing access to more operators from different regions or modes of transport gives the retailer access to markets they could not enter before;
- The retailer is now focused on defining the rules of the network rather than enforcing them. The cost of operating the IT infrastructure is shared between all the network participants (retailer and operators);
- Operators get instant access to ticket revenue and a more accurate settlement process. They're no longer suffering penalty for mistakes made by retailers during the ticket creation or settlement process, if the underlying system behaves correctly;
- Passengers can purchase tickets to travel across multiple transport operators in a single transaction, from a single operator.
- Having a portable ticket wallet that can store tickets for travel across multiple operators greatly simplifies the practicalities of traveling across a multi-operator network. All tickets can be accessed via a single app and be collected using the networks existing infrastructure.

2.7 Distributed Ledger Technology in Mobility as a Service

Default limitations of blockchain implementations for internet of things (IoT) tasks like MaaS are illustrative of some of the pitfalls that are related to early implementations of any DLT, such as latency in transaction confirmation, scalability concerning blockchain size and network expansion, lack of IoT-centric transaction validation rules, the absence of IoT-focused consensus protocols and insecure device integration are required to be addressed before it can be used securely and efficiently in an IoT environment. No doubt there has been a surge in the development of new blockchain platforms including

Ethereum, Hyperledger, Multichain, NEO, and IOTA. However, current blockchain platforms have some distinct weaknesses that forbid an impromptu implementation of the blockchain in an IoT environment [13]. There are a number of blockchain's critical challenges inherent in the IoT, such as:

- Enormous number of nodes and big IoT data;
- Computation: The blockchain activity is unaffordable for the lightweight IoT devices. Some advanced cryptography algorithms used in the privacy-preserving blockchains are too heavy and energy-draining for IoT devices;
- Storage: A massive storage required by Blockchain can be prohibitive for IoT devices;
- Decentralization: Decentralization and heterogeneity are the two major characteristics of IoT [14];
- Unstable and unpredictable connections;
- Latency and capacity: High latency of blockchain is used to ensure consistency in the decentralized blockchain networks.

The blockchain technologies which can potentially address the critical challenges arising from the IoT and hence suit the IoT applications are identified with potential adaptations and enhancements elaborated on the blockchain consensus protocols and data structures. Based on access controls of the blockchain networks, the state-of-the-art blockchains can be categorized into public permissionless, public permissioned, consortium and private permissioned blockchain. [15] (Table in A.1)

2.7.1 Enterprise requirements

Legal requirements that a DLT need to fulfil to be used in the area off payments, clearing and settlement [16]:

- Performance: System needs to be capable of having a high throughput in terms of number of transactions per second (tps) compared to Bitcoin, which currently can only process approximately 7 tps at most;
- Speed. Transactions need to be confirmed and validated in a short time window (preferably milliseconds), compared to Bitcoin and Ethereum, where transactions can take on average 10 minutes and 12 seconds, respectively, to confirm and eventually settle;
- Scalability. System needs to be able to scale immediately as more nodes join the network (latency issues), more transactions are performed (increasing processing power and memory usage required), and the transaction history grows (increasing storage requirements);
- Settlement finality. Legal concept that is mandatory for enterprise applications – once confirmed, transactions cannot be reversed (at least from a legal perspective). This does not apply to public DLTs where settlement finality is only probabilistic: an alternative, longer chain could replace the current chain and reverse all transactions that were previously confirmed.

- Governance. Need for a pre-defined, codified decision-making process involving known, vetted participants, as compared to public blockchains where a social contract exists, and rule changes are achieved through consensus between sometimes anonymous users;
- Privacy/Confidentiality. Transactions or transaction data need to have a certain level of privacy; in public blockchains, all transactions need to be visible to every participant by design.
- Compliance. Participants need to comply with applicable regulatory requirements and the legal framework that they are subject to. This also applies to the network itself and the transactions that take place in the system.
- Safeguard. Need to manually intervene in case an unexpected issue happens (e.g., critical bug). Moreover, anonymous actors with sufficient financial power could initiate a 51% attack against a public blockchain network and reverse transaction history

From this list of requirements, it is clear that it is not possible to legally implement a public blockchain. However, permissioned DLTs in a private network could be able to fulfil these requirements if the network ought to be managed by a trusted entity and every action taken upon the network required permissions. This type of DLT could ensure settlement finality, governance, privacy/confidentiality, compliance and safeguard properties [17]. Regarding performance, speed and scalability, there are also many DLTs which can achieve the required transactional speeds and scalability.

2.7.2 Consensus

Consensus protocols define the law of block generations and/or block selections [15]. They are responsible for the integrity of the information contained in blockchain, and therefore are an essential part of blockchain technology.

There is a spectrum of consensus protocols behind blockchain systems, starting from purely computation bound like PoW to purely communication bound like the Practical Byzantine Fault Tolerance (PBFT), and applied to different Data Models. Depending on the access control policy of the blockchain, the operating environment can be more or less trusted. As such permissioned blockchains usually rely on message-based consensus schema, rather than on hashing procedures like permissionless blockchains. The consensus protocols are the core functions which decide the performance of blockchain-based IoT applications, such as block rates, consistency, scalability and security.

Consensus protocols

- Proof of Work (PoW). A computationally intensive hashing-based mathematical challenge provides a practical means to achieve consensus among the chains of blocks generated in a distributed fashion, while preventing untrustworthy participants from tampering or corrupting the chains. PoW enjoys strong integrity guarantees and tolerates a sheer number of attacks, but this comes at a huge cost: lack of performance [18].

- Proof of X. Participating peers can also be validated via other proofs, instead of finding the nonce, i.e., Proof of Stake (PoS), Proof of Burn (PoB), Proof of Importance (PoI), Proof of Activity (PoA), Proof of Elapsed Time (PoET). The most popular alternative approach to consensus in blockchain is the Proof of Stake (PoS). [19]

- Proof of Stake (PoS). Similar to PoW, it attempts to provide consensus. In the PoS the originator of next block is chosen based on the various randomized combination of minors cryptocurrencies resources and the duration that they hold their resources. Contrary to PoW miners that may not have cryptocurrency and only attempt to maximize profits by increasing computational power, PoS miners defend Blockchain network to protect their wealth and profits. As long as the stake is higher than the transaction fees, participants can trust them to do their job correctly [20]. [21]

- PBFT. Byzantine Fault Tolerance (BFT) [22] is typically used in private blockchain to formulate consensus protocols and guarantees consistency by exploiting the solutions to the Byzantine Generals' Problems [22] – agreement problems. Particularly, the PBFT algorithm [23] has been extensively used to eliminate the Byzantine failures. As a leader-based BFT algorithm, PBFT has one primary and (n-1) backups in an n-node network, where the primary and the backups can be corrupted. The primary is responsible for receiving the requests from clients and initializing the algorithm.

- Tangle. Participants in this network that want to send a transaction, are first required to do a small proof of work, and a weight is assigned to the transaction proportionally to the difficulty of the puzzle that the node has solved for producing it. After which they need to choose and verify two transactions that occurred before. The node is supposed to choose the two transactions to confirm following a weighted random walk, from all transactions it is aware of, and to validate them and all their transitive predecessor transactions. How strongly a new transaction approves its predecessor transactions depends on the weight, i.e., the computational work, that went into producing it. The stability of the system rests on two alternatives: (1) There is a Coordinator that sends trusted transactions to the network and confirms the all transactions in the path selected by the weighted random walk or (2) There is no Coordinator and the majority of the computing power among the nodes behaves correctly. Furthermore, the tangle may contain conflicting transactions since the nodes do not have to achieve consensus on which valid transactions have the right to be in the ledger, meaning all of them can be in the tangle. However, in the case where there are conflicting transactions, there are conflicting parts of the graph, and, therefore, either the transaction sent by the Coordinator will make them invalid or each node will decide on its own which side to trust. This is done through a probabilistic sampling algorithm and deciding according to a statistical test [24]. Currently, since IOTA is a public DLT, to protect the Tangle network against double-spending attacks, the Coordinator is used to confirm transactions. It was IOTA's organization intent to shut down the coordinator or replace it by a set of distributed trusted nodes as the network reached a certain number of confirmed transactions per second [7], however, very recently they have proposed a new consensus algorithm, which will not be addressed in this work. (see Table in

B.1) Further descriptions on the Tangle can be found in Chapter 3.

2.7.3 Data Models

The data model specifies how unit data is stored and how to decide the main ledger. It refers to the case that more than one ledger exists at the same time in a distributed network. If the different ledgers are all accepted, the blockchain network gains more capacity but also the risks to double spending. Some of the most common data models are:

- Chained Blocks. In a blockchain with chained-data structure, such as Bitcoin, only a single chain can be eventually accepted across the system, and the chain is named as the main chain [25]. Each block contains a cryptographic hash of the previous block header, using the SHA-256 hash algorithm, which links the current block to the previous block. This prevents tampering the blockchain [15].

- DAG. The consensus protocol in IOTA, named Tangle [7], uses a Directed Acyclic Graph (DAG) [26] to organize blocks, instead of a chain. In Tangle, a transaction must approve (point to) two previous transactions. Finally, one of the conflicting records can win the approval competition and be accepted. Unlike the single-copy in the chain structure, Tangle does not drop conflicting transactions and keeps them in different branches of DAG. The DAG structure can achieve better capacity [15]. Theoretically, the speed of DAG in IOTA is not limited. Unlike traditional approaches, the more users connect to the network, the faster it works. (see Table in D.1). Further descriptions on the DAG as implemented in Tangle can be found in Chapter 3.

2.7.4 Cryptographic primitives

Virtually all PoW cryptocurrencies are based on hashing algorithms [27], however the choice of hashing algorithm is important, because it determines how the cryptocurrency is mined (what hardware is efficient).

Besides the hash function, the digital signature is another inevitable cryptographic primitive in blockchains. As a basic primitive of cryptography, digital signature is used for ensuring the source authentication [28], source non-repudiation and integrity.

The compatibility of lightweight cryptographic algorithms in blockchains: This is essential for bridging the technologies of the Internet of Things (IoT) and blockchains. According to Wang et al. [29], no existing cryptocurrencies are based on lightweight cryptographic algorithms, except IOTA. (see Table in F.1)

2.7.5 Performance and Scalability

The results show that current blockchain's performance is limited, far below what a state-of-the-art database system can offer [30]. However, some DLTs can still achieve very high throughputs, with negligible differences.

Currently used Proof of Work mechanism is not sustainable in the area of IoT. (see Tables in D.1) Despite of this fact, all the major currencies like Bitcoin and Ethereum and also other blockchain technologies are using this approach. Generally, the PoX consensuses are computationally expensive. According to the latest Bitcoin Energy Consumption Index, Bitcoin miners from all over the world consume over 68TWh of electricity every year to do the proof of work and over 600kWh to complete a single transaction [31]. This is obviously not suitable for IoT scenarios with limited and light-weighted computation. On the other hand, the processed transactions per second (TPS) of the mainstream BC platforms like Bitcoin and Ethereum are very limited, because the single chain of blocks is linear and blocks cannot be created simultaneously. For example, one Bitcoin block takes 10 minutes to be created and added to the main chain, which is very inefficient and fails to meet the requirement of instant transaction in IoT. [32]

BFT-based consensus protocols, such as in Hyperledger, address this performance issues with greater reliability (see Table in C.1) but have poor scalability, as the transaction throughput decreases badly with an increase in the number of validator nodes. The results so far indicate that scaling both the number of clients and number of servers in Hyperledger degrades its performance and even causes Hyperledger to fail. For Hyperledger, having more servers means more messages being exchanged and higher overheads [30].

Another vital consideration is that IoT systems especially the sensors operating in a smart city environment would be generating millions of transactions per day. Therefore, an ideal IoT-oriented Blockchain platform should not have a transaction fee or gas requirement, such as we in do in Ethereum, but not in IOTA or Hyperledger [33].

Regarding IOTA, for the largest TPS, an IOTA stress test held in April 2017 showed that the network had transaction processing capabilities of 112 Confirmed Transactions per Second (CTPS) and 895 TPS within a small test network consisting of 250 nodes [34]. K. Yeow et al. [35] conducted a comprehensive review on decentralized consensus systems for IoT in terms of the data structure, consensus mechanism, and transaction models. From their proposed thematic taxonomy, a synthesized comparison between blockchain-based systems (e.g. Bitcoin and Ethereum) and DAG-based distributed ledgers (e.g. IOTA and Byteball) was conducted. By analyzing and summarizing the pros and cons, the authors found that the DAG outperformed on scalability, transaction confirmation speed and decentralization. They concluded that DAG might be an answer to overcome the challenges of the fast scaling IoT with the need for low latency micro-payments in the M2M P2P decentralized infrastructure [35]. As an example, a new DLT-based charging and billing IoT architecture for electric autonomous vehicles was proposed in [36]. The authors leveraged IOTA based payment system through M2M communication to carry out micro-transactions for charging and billing in electric autonomous vehicles. In another research project, the authors utilized IOTA Tangle to present a streaming data payment protocol which was an application-layer protocol for enabling micropayments among IoT transactions [37]. Another comprehensive study [32] analyzed the IOTA throughput, scalability and decentralization as fundamental metrics for its implementation in IoT like an inter-house energy transaction in a local community. The main findings stated that, in terms of throughput, the nodes scale of network as almost no influence on the transaction speed, in terms of scalability, the number of transactions sent per second and the number of confirmed transactions per second increase almost linearly with the number of senders, and, in terms of decentralization, a permissioned network would allow a guarantee that no one could have above 34% of the network computational power, thus avoiding 34% attacks in case of a coordinator failure and could also avoid parasite chain and large weight attacks when using an intermediary to access the network, taking into account that, in terms of security, the IOTA consensus protocol provides the protection for double spending attacks. However, this study did not assess the computational effort necessary to send a transaction to the Tangle. On the other hand, an evaluation of this operation was conducted on another work [38] which demonstrated that the attachment of transactions to the tangle can be realized with a relatively low delay when compared to blockchains, but that the transactions some transactions took a long time to get confirmed by the Coordinator. None of these works, however, demonstrated results for different hardware types, especially IoT devices, except [39] which have shown not only that the Rasperry Pi's have approximately 20 times less the computational power of an Intel i7 processor when sending a transaction to the tangle for the standard effort defined in the public IOTA network but also that the current battery-powered IoT devices are not suitable for IOTA operations with this defined effort.

2.7.6 IOTA for Mobility as a Service

The most appropriate DLT for MaaS is IOTA. IOTA has been developed specifically for IoT applications involving large, heterogeneous, networks of transacting entities (e.g. sensors or vehicles).

The IOTA Tangle protocol allows for rapid transaction confirmation and, unlike blockchain protocols, validation speeds scale with the size and complexity of the tangle. Another feature of IOTA's Tangle DLT is that the model calls for fee-less transaction recording. Whereas blockchains like Bitcoin must incentivise “miners” to carry out resource-heavy POW-based block validation (and thus transactions incur a fee), IOTA's Tangle model requires no mining, no incentivising of miners and no fees. IOTA's feeless model is especially interesting in IoT deployments like MaaS where micro-transactions between connected objects, vehicles and parties would be the norm. In a blockchain model, fees may outweigh the value of microtransactions and that could limit the scope of potential MaaS transactions.

Speed, scalability and suitability to micro-transactions all make the IOTA DLT model an interesting one for MaaS. But just as with everything else with DLTs, while the proof of concept may be enticing, the ultimate applicability of DAGs like IOTA to real-world applications is uncertain and evolving and will depend on more than the theoretical applicability of the technology to a concrete application. [11].

Chapter 3

IOTA

IOTA is a cryptographic currency, an example of a DLT, that does not use a chain of blocks as the underlying data structure, implementing a structure of a DAG, which calls it Tangle. In the Tangle, each new transaction must validate two previous transactions before can be included in the data structure itself. The nodes select the two previous transactions, which are yet unvalidated and called tips, according to a Markov chain Monte Carlo random algorithm. During the validation, care is taken to ensure that transaction signatures are correct, and that the two validated transactions do not add contradictory information to the tangle. The more new transactions have validated a transaction, directly or indirectly, i.e. through another transaction in the middle, more confident may be the person who added the transaction to the network, that the transaction is valid and will in fact remain in the final tangle. Thus, once a transaction has been directly validated or indirectly by all the tips of the tangle, is considered validated but not confirmed.

IOTA should be particularly scalable compared to chains and would therefore be particularly suitable for Internet applications of the Things, since micropayments can also be implemented in a way sensible because of the non-existent transaction costs. This is particularly relevant for machine-to-machine transactions. Another advantage of the tangle is that the speed with which new transactions are validated increases with the number of transactions added. Currently, however, the network is still in a transitional phase in which, a coordinator, is responsible for confirming transactions. In the next section, we describe IOTA from several technical perspectives.

3.1 Overview

The Tangle runs an asynchronous protocol in a peer-to-peer network to facilitate transaction processing on an immutable, distributed, decentralized ledger secured by cryptographic measures. The Tangle is comprised of transactions, or sites, and edges, which connect sites and form a DAG. The network consists of nodes and each node stores its version of the Tangle. An edge indicates that one site directly approves another. A path symbolizes indirect approval. When a new transaction is issued, it approves two other transactions, which are previous transactions that have never been approved by other transactions. The protocol validates if two approved transactions conflict by examining the Tangle history, and if it discovers a conflict between them, it will not approve those transactions, otherwise a node will store the transaction in its ledger and broadcast it to its neighbors. Each transaction's weight determines its importance in the Tangle. The Tangle defines a transaction's cumulative weight as the sum of the weights of other nodes that directly or indirectly approve the transaction, including itself. In practice, every node's weight is 1. Therefore, the number of nodes that directly or indirectly approve the node represents the cumulative weight. The transactions that are chosen to be approved, are done so using a random walk tip selection algorithm. Currently, the algorithm that biases the random walk is the Markov Chain Monte Carlo (MCMC). In a Markov chain each step does not depend on the previous one, but follows from a rule that is decided in advance, taking into consideration the cumulative weight of the transactions. One of the main concerns of blockchain security is the malicious node issue. Without the use of a tip selection algorithm, like MCMC random walk, nodes can become *lazy* and allow *lazy tips*, which are tip transactions that point to older transactions, to be confirmed. Confirming old transaction is unwanted since it increases the branching factor of the graph and thus, it increases the number of tips. Furthermore, *lazy nodes* do not help the network to grow since no new unapproved transactions get confirmed and can easily allow the occurrence of double-spending attacks on the network, such as the parasite-chain. Despite this, this type of attacks can still occur when using the tip selection algorithm if an attacker can generate enough cumulative weight on his transactions to surpass the cumulative weight of the Tangle (further discussed in Section 3.5). To avoid this, the current IOTA system's stability relies also on a particular type of node, called the coordinator node. The coordinator issues signed zero-valued transactions at a certain rate, which are called milestones, and confirms every transaction in the path of the selected transactions to approve to the latest milestone (at least). The above mentioned tip selection algorithm starts the random walk from a milestone at a certain depth. That said, the nodes validate transactions upon its receipt and during the tip selection. The coordinator confirms transactions by issuing milestones. A transaction is only considered for confirmation if the node has its transaction history path until a milestone and if the node is synchronized with the network i.e. possesses a ledger equal to the other nodes and the latest milestone index.

3.2 Data Models

3.2.1 Ternary

IOTA uses a balanced ternary numeral system. IOTA claims to uses this system because, compared to binary, ternary computing is considered to be more efficient as it can represent data in three states rather then just two. This system is used all through the protocol, from seed and address generation to transaction generation, content and validation. The balance ternary system consists in trits, which represent values of 1, 0, or -1. As such, a tryte, or 3 trits, provide $3^3 = 27$ possible data values.

3.2.2 Genesis, Seed and Address

The entire supply of tokens gets generated in the genesis transaction. This leads to a fixed supply of IOTA-tokens of exactly 2,779,530,283,277,761 [40]. It was chosen for hardware efficiency using the ternary numerical system and in order to be a sufficiently high number to support micropayments in Internet of Things scenarios. The genesis transaction is the only transaction allowed to send tokens from an address with 0 tokens to another address and approve no other transaction. No tokens are created afterwards. As a consequence, the currency is deflationary, because tokens can get lost, e.g. if a user loses their access-key, called "seed". The current implementation uses 81-character seeds composed only of the characters A-Z and the number 9, so 27 possible characters. A transaction is the transfer of tokens from address A to address B, where the number of tokens can be 0, which then the transaction is also called message. An address is like an account that belongs to a seed and that has a 0 or greater balance of tokens. Addresses can be seen as the public half of a public/private key pair, except they are not generated using a public/private key pair algorithm. Addresses are generated using the Kerl [41] hash function, which is based on SHA-3 [42], that starts by generating a private key from a seed, an index, and a security level. The seed and index are added to create a sub seed which is then absorbed and squeezed in a sponge function [43] 27 times for each security level. The result of the sponge function is a private key whose length varies, depending on the security level. To generate an address, the private key is split into 81-tryte segments. Then, each segment is hashed 26 times. A group of 27 hashed segments is called a key fragment, and a private key has one key fragment for each security level. Each key fragment is then hashed once to generate one key digest for each security level. Then, the key digests are combined and hashed once to generate an 81-tryte address. Therefore, the higher the security level defined, the higher the latency to generate a transaction since the security level of an address affects how long the private key is, how secure a spent address is against brute-force attacks, and how many transactions are needed to contain the signature. To transfer IOTA tokens from one address to another, each transaction is signed with the private key to prove to nodes that you own it. As such, addresses can be shared as long as the private key remains in concealment.

3.2.3 Transaction

A transaction is a single transfer instruction that can either withdraw IOTA tokens from an address, deposit them into an address, or have zero-value (contain data, a message, or a signature). There are two basic types of transactions: Input transaction and Output transaction. Input transaction withdraws IOTA tokens from an address of the sender and contains the signature that signs transactions to prove their ownership [44]. In the case of high security level with a very large signature, it is fragmented over zero-value output transactions in the bundle. Output transaction is in charge of depositing IOTA tokens into a recipient's address [44]. If the input absolute value is bigger than output, there will be an extra transaction which will add the remainder to the new addresses of the sender. A valid input transaction must always contain the following: (1) A negative value in the *value* field, (2) An address that contains at least the amount in the *value* field, (3) At least the first fragment of a valid signature is in the *signatureMessageFragment* field and (4) A valid *nonce*. A valid output transaction must always contain the following: (1) A positive value in the *value* field, (2) A valid address and (3) A valid *nonce*. The *nonce* field corresponds to the output of the proof-of-work operation. All transactions must have a transaction hash in the *hash* field, a bundle hash in the *bundle* field, the specified input or output address in the *address* and a trunk transaction and branch transaction field. The trunk transaction is a transaction hash of either an existing transaction in the Tangle or of the transaction with the next index in the bundle. The branch transaction is a transaction hash of an existing transaction in the Tangle.

3.2.4 Bundle

A bundle is a group of one or multiple related transactions, that rely on each other's validity. It acts as a transactions container to transfer data or tokens. It is always an atomic operation, i.e., either all transactions are successful or none. The structure of a bundle consists of output, input, and remainder transactions, which are called head, body, and tail, respectively. Typically, the tail has index 0, and the head is the last transaction in the bundle. All transactions, starting from the tail, are connected by reference to the one with the next index. These connections allow nodes to reconstruct bundles and validate their contents. In order to attach transactions to the ledger, the head transaction needs to be connected to one tip and the other transactions in the bundle to another tip, as seen in Figure 3.1.

3.3 Network

According to the access and permission control, networks can be divided into private and public networks. The public IOTA network is usually considered as the Mainnet responsible for processing all IOTA cryptocurrency transactions. There are no access controls for participants to join the network so that anyone can run a node to read from and write into the public ledger. The private network generally requires a permission management mechanism, which the IOTA does not offer (e.g., membership service provider in HyperLedger Fabric network [45]) leaving its management entirely to whoever decides to implement it.

3.3.1 Node

Nodes are the core of an IOTA network. They run the node software that gives them read and write access to the Tangle. Like any distributed system, nodes are connected to others called neighbors to form an IOTA network. Nodes are responsible for the following key functions: (1) Keeping a record of the addresses with a balance greater than 0, (2) Validating transactions and (3) Attaching valid transactions to the Tangle. When one node receives new transactions, it will try to validate them in the Tangle to make sure that their histories do not conflict and that counterfeit transactions are never confirmed, and if validated, forwards the transactions to all its neighbors. This way, all nodes eventually validate all transactions and store them in their local copy of the Tangle called a ledger. A node is synchronized when it has solidified all milestones up to the latest one. Solidification is the process in which a node asks

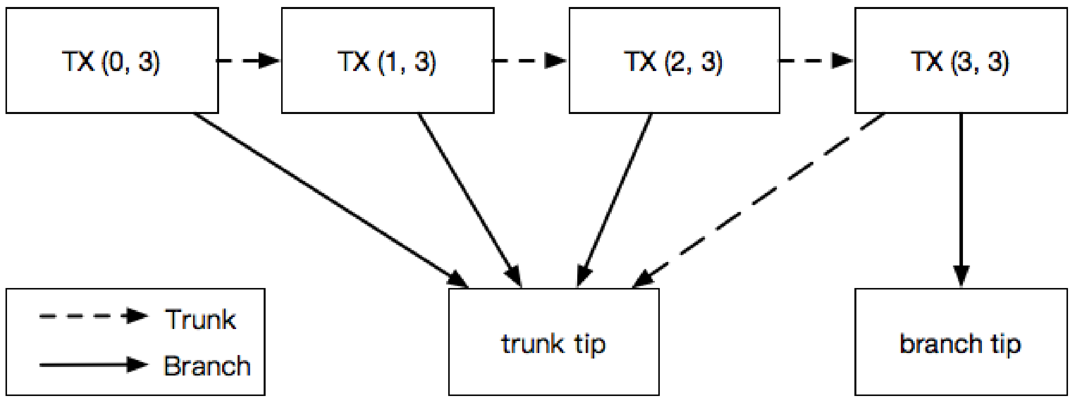


Figure 3.1: Bundle structure

its neighbors for the history of all milestones in the Tangle, starting from an entry point milestone and ending at the latest one. New transactions can only be considered confirmed when solid and directly or indirectly referenced by a transaction that is signed by the Coordinator, therefore a node must be synchronized.

3.3.2 Coordinator

The coordinator is an application that is run by the IOTA foundation in the Mainnet and whose purpose is protect the Tangle from attacks such as parasite chains. Nodes use the coordinator to reach a consensus on which transactions are confirmed. Every node in the same IOTA network is hard-coded with the address of a coordinator. So, whenever nodes see a milestone, they make sure it's valid by doing a number of checks, including: (1) The milestone came from the coordinator's address and (2) The milestone doesn't reference any invalid transactions. The coordinator's address is derived from a Merkle tree [46], where the address is the root, and the private keys are the leaves. The private keys are used to sign milestones before sending. As a result, if the coordinator were to ever send an invalid milestone such as one that references counterfeit transactions, the rest of nodes would not accept it. When a transaction in a valid milestone references an existing transaction in the Tangle, nodes mark the state of that existing transaction and its entire history as confirmed.

3.4 Operations

3.4.1 Tip Selection

In IOTA Tangle, the number of tips increases as new transactions keep adding to the ledger. It becomes crucial how to wisely select two tips from all the valid tips so that the whole Tangle can grow in an ”healthy” way.

EntryPoint

There must be a starting position for any walk to start the random walking. It is easy to think of taking the genesis node as the entry point. However, this will bring obvious efficiency decrease while traversing a long path when the graph grows large. Currently, a walker starts at a specified n depth number of milestones in the past. The DAG starting from this old Milestone is called subgraph. Moreover, this old milestone located at n depth is the entry point. Based on the experimental results [47], it was concluded that any starting position placed further than $10\lambda - 20\lambda$, being λ the transaction arrival rate, provide the same growth of number of tips, meaning that the starting position does not affect the selection mechanism. It was also desmonstrated that the average number of tips seem to grow linearly with λ (transaction arrival rate), meaning it is scalable.

Cumulative Weight

For each transaction in the subgraph, there is a property called cumulative weight, defined as the summation of its weight and the sum of weights of all transactions that directly or indirectly refer to this transaction. The weight of a transaction reflects the computation resource that the sending node puts into this transaction. As a new transaction comes to the tangle, all previous transactions connected to it in the subgraph updates by adding the new weight. For example, in the lower part of Figure 3.2, every transaction connected to X increases its weight by 3 after X is attached. However, updating the cumulative weight too frequently will decrease the efficiency.

MCMC Weighted Random Walk

Having the entry point and current Tangle state of cumulative weights for all transactions, the walker walks through the subgraph twice, for each time from the entry point to reach a tip, to get two selected tips. For example, the selected tips in Figure 3.2 are v9 and v8, with the random walk paths $(v1, v3, v6, v7, v9)$ and $(v1, v2, v4, v5, v8)$, respectively. Specifically, the walk chooses next approver transaction in a probability determined by the cumulative weights and a randomness parameter, called α. The tip selection algorithm never ends inside a bundle even in the case that the rest of the bundle has not yet arrived at that node. If the selected tip is not a tail in a bundle, the MCMC will “walk back” to the previously visited tail transaction.

On a side note, the bundled transactions might by chance not be approved by other transactions on the Tangle. In this case the node has two options: Either it reattaches the transaction or it promotes it. Reattaching/ Replaying means issuing the same transaction again with a different hash, because different trunk and branch-transactions are selected, meaning to redo the PoW for newly selected tips. This might be necessary if the issued transaction does not get approved by enough transactions to get confirmed, so the user attaches it to the Tangle in another spot. The previously attached transaction is left behind and gets orphaned.

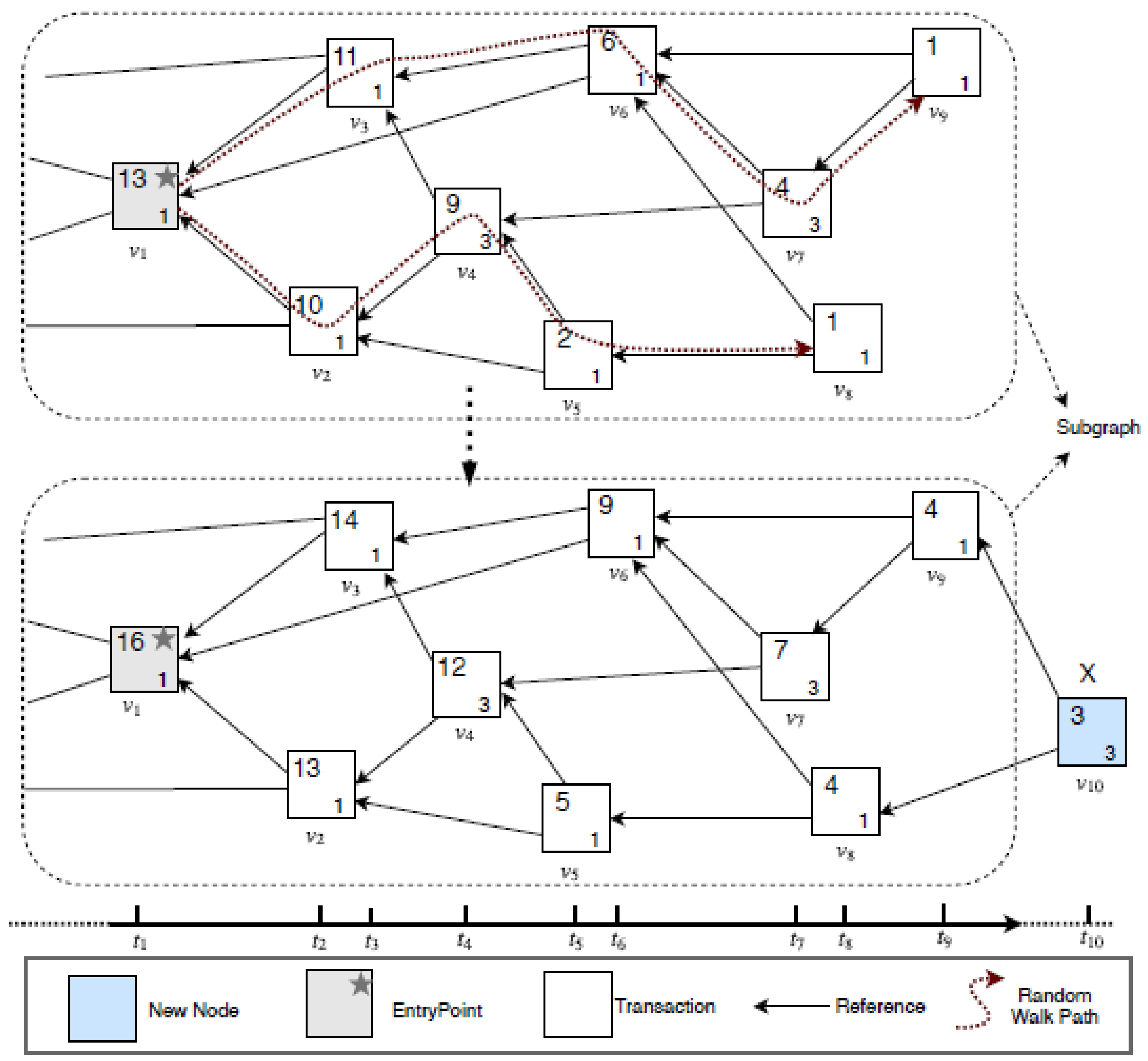


Figure 3.2: MCMC weighted random walk

3.4.2 Proof-of-work

A proof of work (PoW) is a piece of data that is calculated using trial and error to meet certain requirements. As a spam prevention measure such as Hashcash [48], each transaction must include a PoW to be valid. This PoW can be difficult to do, depending on the chosen difficulty level in the network, but is always easy for nodes to validate. To calculate the PoW for a transaction, the values of all the transaction fields are converted to trits and hashed, using the Curl hash function. This process continues until the transaction hash ends in the same number of 0 trits as the minimum weight magnitude. Whenever the transaction hash does not end in the correct number of 0 trits, the value of the transaction's nonce field is incremented and the transaction hash is hashed again. It is possible to compute the PoW locally or to outsource its computation.

3.4.3 Validation

Transaction Validation

Transaction validation is accomplished in two separated steps: (1) When a node performs tip selection (weighted random walk) for a requested transaction (Bundle validator and Ledger validator) and (2) When a node receives a transaction (Transaction validator). The transaction validator checks the following: (a) Proof of work is done according to the minimum weight magnitude, (b) The value of any transaction in the bundle does not exceed the total global supply of tokens, (c) All IOTA tokens that are withdrawn in a bundle are deposited in remainder addresses and (d) Any signatures in value transactions are valid. The bundle and ledger validators check the following, during the weighted random walk: (a) The value of any transaction in the bundle does not exceed the total global supply of tokens, (b) All IOTA tokens that are withdrawn in a bundle are deposited in remainder addresses, (c) Any signatures in value transactions are valid and (d) Each bundle does not lead to a double-spend by checking the values of all addresses in a bundle. If a double spend is found, the weighted random walk steps back one transaction and finds another route to a tip transaction.

Transaction Confirmation

After a transaction is sent to the Tangle, its cumulative weight will increase as time goes by according to the MCMC. Thus, more and more new transactions will tend to validate it directly or indirectly. In the mean time, the coordinator allows transactions to go from validated to confirmed, thus making the associated addresses with the correct balance. It does so by regularly sending bundles (milestones) that reference and approve two transactions selected by the MCMC. Therefore, the higher the cumulative weight of transaction, the higher its probability of getting confirmed.

3.4.4 Transaction Processing

The process flow to complete a transaction is described below:

1. Generate Bundle Hash: During preparation of transactions, Kerl hash function with sponge constructor absorbs all transactions objects necessary for validation (address, value, obsolete tag, timestamp, current index, and last index) and squeezes the hash.

2. Sign Input Transactions: All hashing in this step uses the Kerl hash function. There are two main sub-steps to transaction signing: (1) Generating the private key and (2) Signing the transaction. First, a subseed is generated from the user's seed by taking the Kerl function to ensure it is independent enough from the original seed. Then, the private key can be obtained from a key generator by successively hashing this subseed 27 times per security level. Second, one can use signature fragment generator with a private key and the above-mentioned bundle hash to get the transaction signature. There are two notes to take into account: (1) Input transactions should not reuse the same private key since it would disclosure part of it and (2) There are in total 81 trytes in a bundle hash. Depending on the predefined security level value (S = 1, 2 or 3), the client will take the first

S x 27 (i.e., 27, 54, or 81) trytes, respectively, of the bundle hash, combined with the private key to generate a transaction signature for each transaction. Since the signature is hashed 27 times per security level, it results in a key length of S x 27 x 81 trytes, i.e., 2187, 4374, or 6561 trytes, respectively. Each 81-tryte hash generated constitutes a key fragment in a transaction. After this step, all signatures are filled in the corresponding transactions.

3. Tip Selection: MCMC is used to randomly select two unvalidated transactions. When the client is requesting transactions to approve, he can specify how many milestones in the past the node starts the tip selection algorithm. The greater the depth, the farther back in the Tangle the IOTA node starts. A greater depth increases the time that nodes take to complete tip selection, making them use more computational power.

4. Proof of Work: For each transaction included in the bundle, the issuing node must do PoW, in order to have the proposed transactions accepted by the network, according to the pre-set PoW difficulty level called minimum weight magnitude. This difficulty level is defined on the nodes network configuration and is recommended to be the same for all the network.

5. Validation: Transaction validation and confirmation is required.

3.5 Security Analysis

For the designers of a cryptographic system it is crucial to understand all possible attack vectors and the ways to deflect them. The double-spending attacks described in the following give only a short insight into possible scenarios. A double-spending attack is an attack in which a malicious user attempts to spend the same digital currency multiple times. This is a problem that also affects Bitcoin and other cryptocurrencies. It is of paramount importance that the probability of a successful double spend in a cryptocurrency is almost zero. Otherwise the validity of transactions cannot be guaranteed, which will make the cryptocurrency worthless. There are multiple ways to attempt a double-spending attack in IOTA. A few of them are described in the white paper [7]. These are the large weight / outpace attack [7] [49], the splitting attack [7] and the parasite chain attack [7] [50]. The following does not attempt to be a comprehensive list. There are other possible attacks, e.g., replay attacks [51], 34% attack [52], Sybil Attack [7].

3.5.1 Large Weight / Outpace Attack

The attacker tries to issue a double-spend transaction while forking off the originally accepted transaction. Two goals have to be attained: First, the merchant must accept the original transaction and deliver the purchased goods, and second, the network must be convinced to build upon the attacker's sub-tangle. The way to achieve these goals is to wait long enough for the merchant to accept the payment. During this time, the attacker can build his sub-Tangle offline. Furthermore, the attacker must outpace the growth of cumulative weight in the main-Tangle by doing excessive PoW on the sub-Tangle. Outpacing the main-Tangle would be easier if the attacker could issue just one transaction of large weight. [7] That is why the own weight of a transaction is limited. Consequently, the attacker must issue numerous transactions on the sub-Tangle which reference only transactions in the sub-tangle, but not in the main-tangle. Eventually, for the attacker it boils down to having more than a third, i.e. 34%, of the total hashing power to force the enough walkers from the MCMC algorithm into walking along his Tangle. Described with the Byzantine Generals Problem [22] this means that more than a third of the generals are traitors, which results in a faulty. system.

3.5.2 Splitting Attack

The Splitting Attack is very similar to a large weight attack, except that the attacker does not need to run a race against the computing power of the entire network all by himself. Instead, he tries to balance the weights of the two emerging sub-Tangles, so that half of the network grows one sub-Tangle, and the other half the other. The attacker issues two or more non-conformant transactions, thus splitting the network. Without these conflicting transactions some node would eventually approve one transaction of either sub-tangle and therefore merge them together. The more balanced the two sub-tangles grow by themselves, the less work the attacker has to do, because the network works on the two sub-tangles equally. This lets them grow long enough that a merchant would accept the attacker's payment. His benefit is that he can spend his funds on both sub-tangles, resulting in a double-spend.

To prevent this attack from being successful, the honest nodes have to make it too difficult for an attacker to balance the two sub-tangles. To achieve this, the MCMC is designed in a certain way. First of all, the entry point of the random walk must start deep enough in the Tangle to make sure it starts before the split. Otherwise, the walkers would not be able to make a decision between the two sub-Tangles. And secondly, the algorithm must decide sharply, i.e. rather deterministically, for the sub- Tangle with greater total weight. This second point can be achieved by making the transition probability more dependent on α. [7] A high α value is needed for weighted random walk tips selection. So that it will be very hard for an attacker to keep the balance between two branches. However, the higher α is, more tips could be left behind.

A factor making it even harder for the attacker to accurately balance the sub-Tangles is network latency. Transactions take time to propagate through the mesh network all the way to servers owned by the attacker and vice-versa. This forbids any node to be omni-present so that an attacker could not react on changes of the two sub-tangles quickly enough.

3.5.3 Parasite Chain Attack

The parasite chain attack equals an extended large weight attack. The attacker tries to build a side-tangle quickly to issue a double-spend transaction. At first, by issuing a legitimate transaction on the main Tangle, he pays a merchant and receives the corresponding asset. Afterwards, he is then able to make the previous transaction obsolete by constructing a side- Tangle which must eventually become the main tangle. The difference here is that the side-Tangle builds upon the main Tangle by occasionally referencing it. This allows it to reach even higher score and height. Therefore, it is called a parasite.

This attack can be prevented when nodes use some type of MCMC tip selection strategy if we assume that the main Tangle has more hashing power than the attacker. Consequently, the main-tangle receives more cumulative weight than the side-tangle, which makes walkers of the MCMC more likely to walk through the main- tangle. After the legitimate transaction on the main-tangle, no transactions approve tips from both Tangles at once, because the legitimate transaction conflicts with the double-spend transaction on the side-tangle. From this point on, it is the same as a simple large weight attack, except that the side-tangle has a slightly higher probability in the MCMC. [7]

Chapter 4

Solution

Following the descriptions and analysis conducted in Chapter 2 and Chapter 3, IOTA 1.0 still incurs several limitations in terms of security and availability, and its performance, throughput and scalability results are inconclusive for designing a feasible solution in transport mobility. Despite all its limitations, IOTA is still the most promising. The main goal of this solution is to create an immutable ticketing system as a proof of concept using IOTA, and allow further benchmarks. This system will allow users to buy and use transport tickets for many service providers (e.g., Metro, Taxi, and Bus) while not requiring an entity to validate and distribute the revenue of each of these utilizations. The main focus is on creating and implementing models that will support the use of IOTA technology for many different use cases, depending on the transport operator, and not so much on the purchase of tickets or audit operations.

This Chapter provides a conceptual, high-level overview of the system developed in this dissertation's work. In the next section 4.1, the main components of the solution and how they relate with each other are introduced. The solution's architecture is described in more detail in section 4.2 along with the introduction of the different transport models and a description of the IOTA network. Then, we present the core of the solution (Wallet Manager API) in Section 4.2.3, from its importance to its usage. In Section 4.2.4, we describe two multiplatform frontend applications for the interaction with the system. In Section 4.3, the trust and threat models that were considered are defined. Later, an overlayer of the main solution is presented in Section 4.4, to solve some of the problems identified previously. Finally, the use cases chosen to implement, including their main features and how they possibilitate the use of IOTA in the public transport scenario in section 4.5.

4.1 Overview

The solution is divided into three parts: Frontend (client and service provider applications), Wallet Manager API, and IOTA, as shown in Figure 4.1.

The system's frontend is a multiplatform application for the client and service provider, which is responsible for interfacing requests to the wallet manager or the IOTA network. A client represents the end-user of the ticketing system, while a service provider represents a transport operator. On the

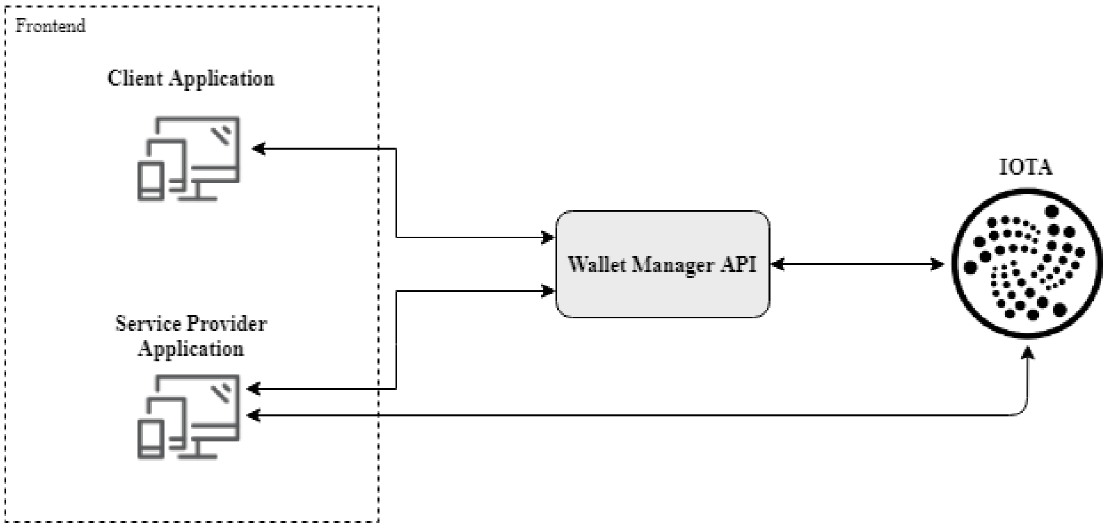


Figure 4.1: Solution's High-Level Architecture

client application, each client is able to buy IOTA tokens and request tickets or a balance update. On the service provider application, each provider can update its price, receiving address or balance, and receive tickets from clients and forward them to the IOTA network. The wallet manager is the backend of the system. It is in charge of receiving requests from the frontend and performing the requested operations while handling requests to the IOTA network when necessary. The latter part runs the IOTA Tangle protocol and is responsible for processing all the incoming requests from the wallet manager or the frontend.

Besides choosing IOTA as the core DLT for this project, the primary design decision was made to consider its flaws and benefits. We will consider, in the course of this work, as possible double-spending, the scenario where a client issues a fake ticket and is able to use a transport, even though the ticket contained in the transaction later becomes detected as a double-spend in the Tangle. Mainly, the solution could fall into two categories: (1) A permissionless solution where each client would possess a virtual wallet able to generate transactions directly to the IOTA network and (2) A permissioned solution with an intermediary component able to generate transactions on behalf of the client, directly to the IOTA network. The first solution's main benefit is distributing the computation per client. In opposition, the latter required the computation to fall into IOTA nodes, which ought to be managed by a single entity, making it a more centralized solution. In terms of computational resources required, the results from studies in chapter 2 are inconclusive for IOTA nodes, as further benchmarks with more parameters changes were required, however, it is clear that mobile devices may not be powerful enough to complete operations such as proof of-work in satisfactory amount of time. Regarding security, (1) would permit anyone to interact and tamper with the network and there was no double-spending control and with (2) only registered users could interact with the network through an intermediary (wallet manager or service provider endpoint) and an anti-tampering and anti-double-spending mechanism could exist ensuring that 1) transactions were generated by a trustful entity, 2) transactions generated would not be tampered with and 3) transactions could not be sent twice. The security mechanism was thought but not implemented, as first it was required to ensure IOTA nodes could handle the necessary number of transactions of a state-of-art public transport ticketing system, and for this the requirements were more focused to traditional use cases. The next sections will describe, in further detail, each mentioned component above as well as the detailed architecture proposed based on some of the results obtained in chapter 6.

4.2 Architecture

This section presents a more detailed solution's architecture, illustrated in Figure 4.2 and description of the solution, explaining in detail how each component works. First, in Section 4.2.1, an application of common transportation models using IOTA is described. Then, in 4.2.2, we will detail how the IOTA network was designed. Later, in Section 4.2.3, the thought process behind Wallet Manager API is described and, finally, in Section 4.2.4 both the Client Application and Service Provider Application are detailed.

To better understand this architecture, the process where a client requests and uses a ticket is simplified and illustrated in Figure 4.3. According to the functionalities and use-cases were defined specific requirements. For this scenario, the client is already registered in the wallet manager and has enough tokens to buy tickets for a generic transport option. After being authenticated in the application, the client can start the process of getting a ticket and using it. To do so, he has to request it to the wallet manager, which will generate a request of a bundle to an IOTA node on behalf of the client and containing the price for the trip. The node will process the request and respond with the non finalized bundle. The bundle is then sent to the client with some more parameters which will guarantee the authenticity of the bundle, forming a ticket. Then, the client sends the ticket to the service provider endpoint that validates the authenticity of the ticket and sends the bundle to the proxy. This service provider endpoint is implemented in the service provider application in order to easily simulate a transport gateway/terminal. The proxy is then responsible for computing the proof of work, finalizing the bundle and sending it to the node, when available. Finally, the node will receive the ticket, validate the bundle, store it and broadcast it to its neighbors. In the meantime, the coordinator is generating and issuing milestones to every other node, confirming transactions.

4.2.1 Transportation Models

In the most common transportation models there is a ticket validation component at the entrance of the service provider gateway/terminal/vehicle. This component's behavior is defined by the service provider endpoint. To do this, it is necessary for the client application to be able to interact with the service provider endpoint and the latter with a node or proxy in the network. Since it is through the client application that the tickets are sent, it requires knowledge of the recipient and the value to be transferred. These parameters will be communicated by the wallet manager in the form of a ticket. Therefore, each service provider must be registered in the wallet manager and have a price/rate set. The next sub sections will describe state of the art transportation models using IOTA technology.

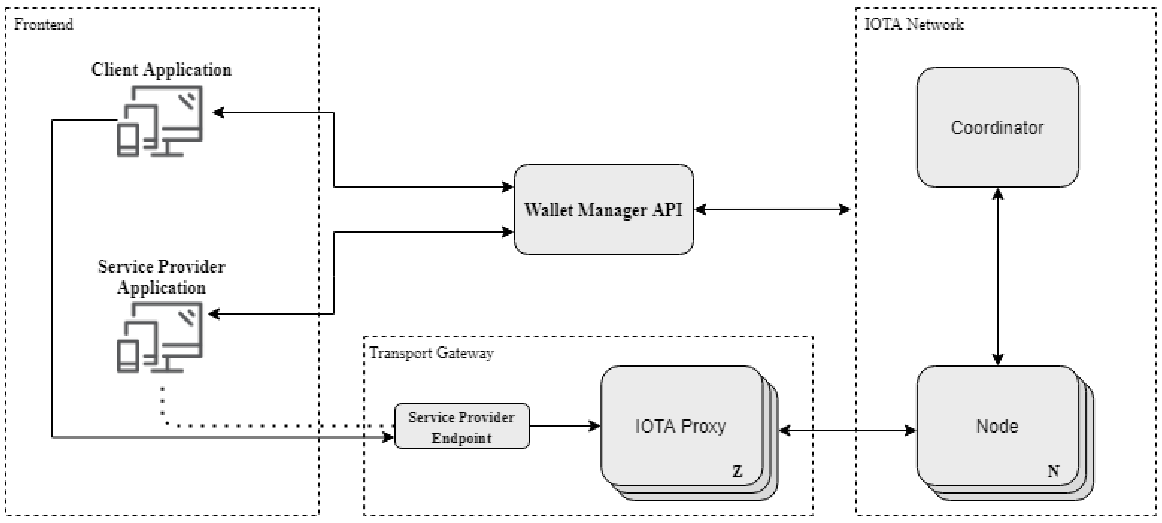


Figure 4.2: Solution's detailed Architecture

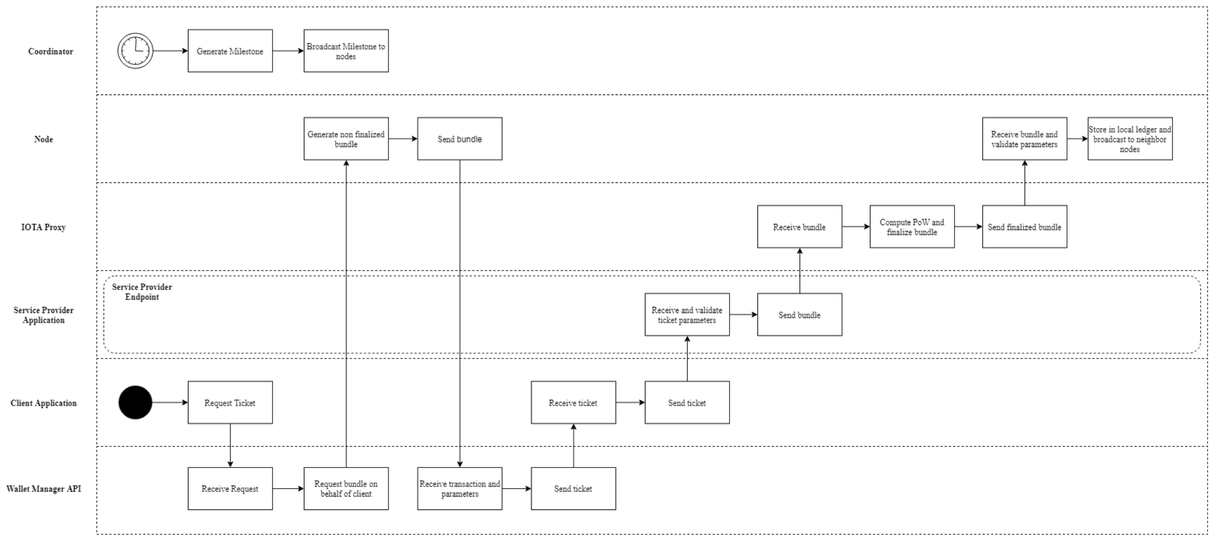

Coordinator
Generate Milestone
Broadcast Milestone to nodes
Node
Generate non finalized bundle
Send bundle
Receive bundle and validate parameters
Store in local ledger and broadcast to neighbor nodes
IOTA Proxy
Receive bundle
Compute PoW and finalize bundle
Send finalized bundle
Service Provider Application
Service Provider Endpoint
Receive and validate ticket parameters
Send bundle
Client Application
Request Ticket
Receive ticket
Send ticket
Wallet Manager API
Receive Request
Request bundle on behalf of client
Receive transaction and parameters
Send ticket

Figure 4.3: High-level process description of a client's ticket request and utilization

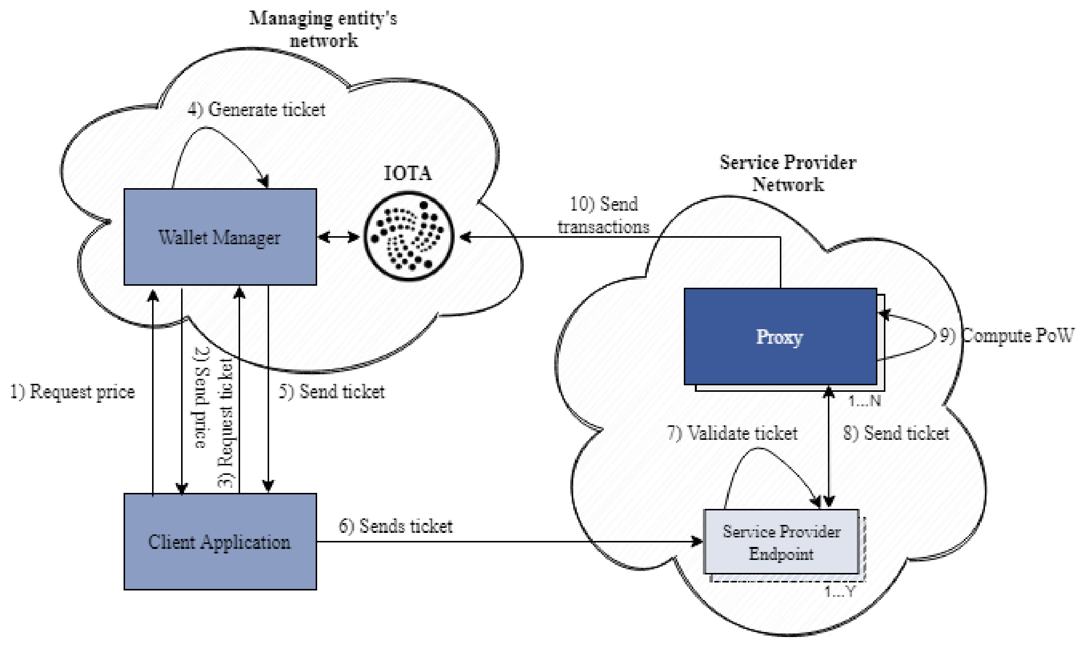


Figure 4.4: Traditional model diagram

Traditional

In the traditional model (e.g., Metro and Bus) there is a ticket validation component at the entrance of the service provider transportation method composed of multiple gateways/terminals. This model requires a ticket to be sent by the client in order to become available to use. The ticket price is fixed and is set using the service provider application. An high-level usability scheme can be observed in Figure 4.4.

Taxi

The taxi model (e.g., Taxi and Uber) specified the necessity of providing a ticket cost based on the distance to travel and service provider price rate. It is similar to the traditional model, with the exception that the ticket cost isn't fixed and the service provider endpoint represents the driver's application or a single gateway/terminal installed on the transportation vehicle. An high-level usability scheme of this can be observed in Figure 4.5.

Wallet Consumption

The wallet consumption model (e.g., electrical scooters and bicycles) is based on the client's usage time of the service, thus requiring check-in and check-out transactions. This usage time is registered on the service provider endpoint, which is, in this case, the transportation vehicle software being used. A check-in transaction represents a zero-valued transaction to symbolize that that specific user has started to used the vehicle. In order to stop using it, the client has to request a check-out, which in turn will cause the endpoint to send a transaction to a proxy/node with the value of the trip. The value of the trip is calculated based on the difference of the trip end time and start time, taking into a account a certain rate per unit of time. An high-level usability scheme of this can be observed in Figure 4.6.

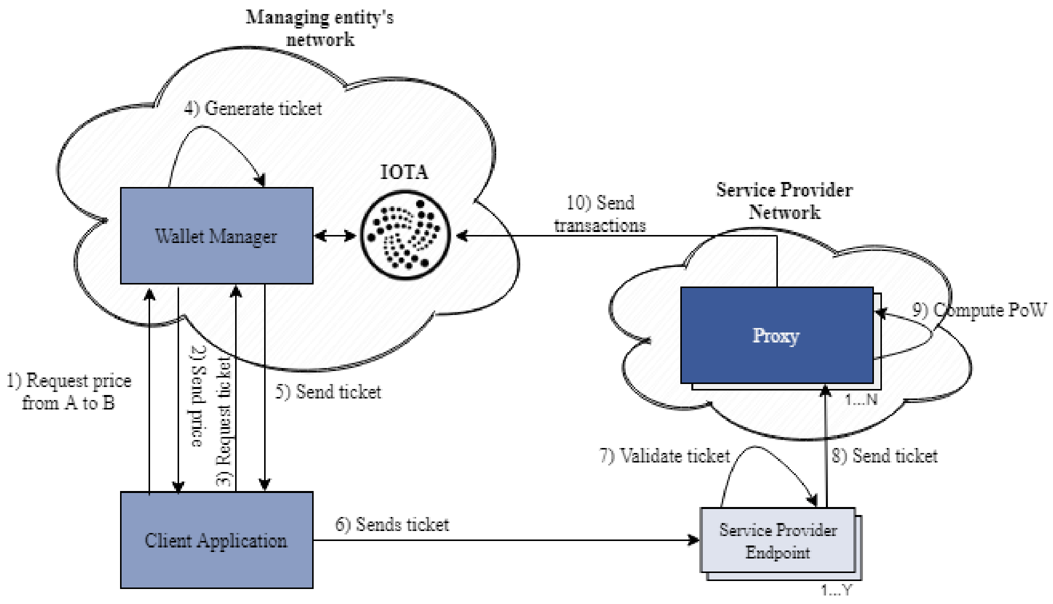


Figure 4.5: Taxi model diagram

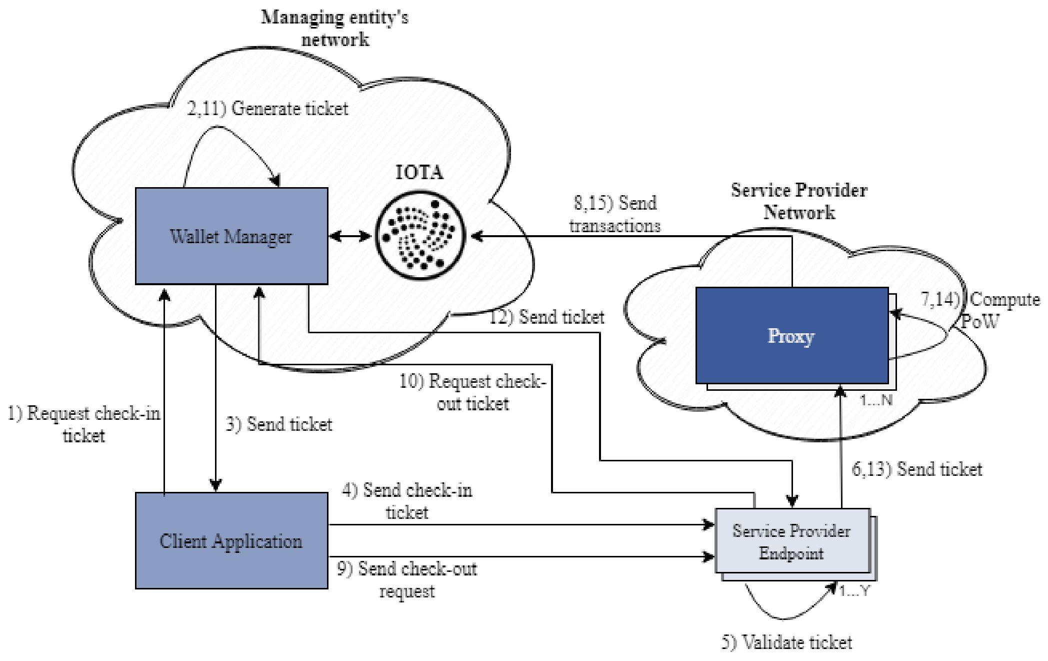


Figure 4.6: Wallet Consumption model diagram

4.2.2 IOTA

Assumptions and Limitations

The design of the IOTA network for this system had to take into account some assumptions and some limitations. As assumptions, first, that the whole network would be managed by a trusted party and thus avoiding 34% attacks on the network even on the absence of the coordinator, and possibly parasite chain and large weight attacks, and second, that whatever IOTA component (proxy or node) handling the connections from the service provider transport gateway/terminal had to eventually be able to connect to the main network and broadcast transactions. As limitations, first, the coordinator is a single point of failure of the main network and without it no transaction gets confirmed and second, the performance of a node and proxy when handling client requests was inconclusive due to a lack of metrics gathered in previous studies, which had to be further analysed in Chapter 6. In the next sub sections, a description of each network component and a justification for the chosen design can be found.

Components Description

IOTA is a DLT that allows participants in the network to transfer immutable data and value (tokens) among each other, called transactions. Before sending a transaction, one is required to do a small proof of work, and a weight is assigned to the transaction proportionally to the difficulty of the puzzle that the node has solved for producing it. There are three ways to compute the proof of work: (1) Locally on the machine that is generating the transaction, (2) Remotely on an IOTA node or (3) Outsource to an IOTA proxy. The first one is out of the scope of this project as seen previously due to performance and security implications.

An IOTA coordinator is one of the core components of the network whose purpose is protect the Tangle from attacks such as parasite chains. Nodes use the Coordinator to reach a consensus on which transactions are confirmed. It's the coordinator that allows your transactions to go from validated to confirmed, thus making the associated addresses with the correct balance. It does so by regularly sending bundles (milestones) that reference and approve two new random transactions in the ledger, which later on will be validated by the nodes. Nodes have to make sure that: (1) The milestone came from the Coordinator's address and (2) The milestone doesn't reference any invalid transactions. The coordinator's address is derived from a Merkle tree, where the address is the root, and the private keys are the leaves, which is a way to prove to nodes that it owns an address without signing transactions with the same private key every time. Every node in the same IOTA network is hard-coded with the address of a coordinator. This way, to create a network with more than one coordinator: (1) All coordinators had to share the same merkle tree or all nodes should be allowed to support more than one coordinator address, implying that every coordinator would have a different address, (2) Nodes should support trusted and dynamic coordinator's, so that coordinators could be added and removed from the network when needed and (3) A mechanism to reach consensus for multiple coordinators or a cluster of multiple coordinators should exist. It was IOTA's organization intent to replace the coordinator for another consensus mechanism, however this was not achieved during the lifespan of this project, leaving no other option as to use a single point of failure coordinator. It is, however, discussed in Section 4.4.1 an alternative to one single coordinator consisting in clustered coordinators, removing the single point of failure from the system and reducing the risk of centralization.

An IOTA node is the other core component of the network. Nodes are connected to others called neighbors to form the network. When one node, receives a transaction, it will try to validate it and forward it to all its neighbors. This way, all nodes eventually validate all transactions and store them in their local copy of the Tangle. Nodes are responsible for the following key functions: (1) Keeping a record of the addresses with a balance greater than 0, (2) Validating transactions, (3) Attaching valid transactions to the Tangle, (4) Synchronizing with other nodes and (5) Performing requested operations through the API. The process of validating a transaction is to make sure that their histories do not conflict and that counterfeit transactions are never confirmed. Transaction validation is accomplished in two separated steps: (1) When a node performs tip selection (weighted random walk) for a requested transaction (Bundle validator and Ledger validator) and (2) When a node receives a transaction (Transaction validator). The transaction validator checks the following: (a) Proof of work is done according to the minimum weight magnitude, (b) The value of any transaction in the bundle does not exceed the total global supply of tokens, (c) All IOTA tokens that are withdrawn in a bundle are deposited in remainder addresses and (d) Any signatures in value transactions are valid. The bundle and ledger validators check the following, during the weighted random walk: (a) The value of any transaction in the bundle does not exceed the total global supply of tokens, (b) All IOTA tokens that are withdrawn in a bundle are deposited in remainder addresses, (c) Any signatures in value transactions are valid and (d) Each bundle does not lead to a double-spend by checking the values of all addresses in a bundle. If a double spend is found, the weighted random walk steps back one transaction and finds another route to a tip transaction.

An IOTA proxy is solely dedicated to performing proof of work and proxying the other client's requests to the node.

Network

Considering the functioning and importance of the node and the proxy, it is theoretically possible to assume that the node has a greater impact on the system's network and that a machine would have less free computational resources running the node software when compared to a proxy. As such, a proxy would apparently be more suited for outsourced proof of work computations. This was further proved in the benchmarks presented in the Chapter 6. As such, an IOTA proxy reveals easy to manage, repair and scale, and better performing when handling client requests making it suited to handle communications from transport gateways/terminals. This way, proxies handle this requests and compute the proof of work for the to be sent transactions while nodes have more computational resources to perform other tasks. It results in a need for more proxies rather than nodes, as the proof of work operation is the most intensive task to be performed in IOTA software, as show in Chapter 2 and 6. With this results in mind, the design of the network consists in a coordinator, a node per service provider and a proxy per service provider transport gateway/terminal, which could be increased as further needed.

4.2.3 Wallet Manager API

The wallet manager serves the client and service provider applications. This component acts an intermediary between the users and IOTA network, responsible for keeping client seeds, service provider seeds, addresses and prices, and performing operations such as balance updates, token transfers and ticket generation. Besides providing an API for both the client and service provider application, it uses the IOTA IRI API which is the node's API to interact with the network. The implementation details as well as the operations descriptions will be detailed in section 5.3.2. In terms of design, it was never considered to be a fully scalable and fault tolerant component but more of a proof of concept that could process asynchronous requests for a small group size of users, with the objective of proving it is possible to create a ticketing system using IOTA technology.

4.2.4 Frontend

The frontend consists in a single multi-platform application whose intent is to provide an interface to use the Wallet Manager API (in most cases) and it has different functionalities for both the client and service provider. To better crease its differences, it is abstractively split into client application and service provider application.

Client Application

The client application should be used by the clients of the system (i.e., a person who wants to buy and/or use tickets to ride a form of public transport). Its main functionalities consist in: (1) Balance and transaction history update, (2) Token acquisition and (3) Different ticket generation alternatives. The implementation details as well as the operations descriptions will be explained in section 5.4.

Service Provider Application

The service provider application should be used by the service providers of the system (i.e., Taxi, Metro, Bus). Its main functionalities consist in: (1) Balance and transaction history update, (2) Address update and (3) Price update. It is important to note that the service provider endpoint, shown in the Figure 4.2, was implemented in the service provider application in order to execute a more agile proof of concept. Service provider's endpoint is able to: (1) Receive tickets and (2) Send transactions. The implementation details as well as the operations descriptions will be explained in section 5.4.

4.3 Models

This section describes the models considered during the design of the solution. A definition of the solution's trust model can be found in Section 4.3.1, where the considered assumptions are stated. Then, possible threats are listed in Section 4.3.2.

4.3.1 Trust Model

Each component described in the system's architecture runs in distinct environments: the client application executes in the user's mobile, the service provider application executes in a service provider's device, the wallet manager and IOTA network run in the main entity's network and finally the IOTA proxy and service provider endpoint are located in each service provider's network, as depicted in Figure 4.7. To better understand these different environments and how these components should work, it is necessary to establish a satisfactory trust model. The trust model helps us recognize the special characteristics of the system and how the various entities are expected to behave. The following analyzes the assumptions taken into account to define the trust model of the proposed solution:

- the wallet manager is trustworthy: the software and hardware which forms the wallet manager is reliable and not compromised, usernames and seeds are unique and securely stored, no data is tampered with;
- the IOTA network is trustworthy: the software and hardware which runs the coordinator and nodes is reliable and not compromised, the coordinator will send valid milestones and the nodes will not tamper with the received data. The used cryptographic algorithms are considered secure and not broken; Therefore, the managing entity and its network is trustworthy.
- the service provider endpoint is trustworthy: ticket's data is properly validated and real transactions are sent to the proxy. Its software and hardware are reliable and not compromised;
- the proxy is trustworthy: it does not tamper with the requests sent by the endpoints and always proxies them, and computes valid proof of works. Its software and hardware are reliable and not compromised; Therefore, every service provider's network is trustworthy.
- the service provider's application is trustworthy: service providers software and hardware are reliable and not compromised;
- the client's application can act maliciously: clients can try to get access to other client's accounts, send fake tickets or reuse tickets (double-spending);

4.3.2 Threat Model

A threat model identifies potential threats and vulnerabilities of the system. Its analysis provides a way to design better defenses against attackers and possible countermeasures, increasing the security of the system. From the previous section, only the client's application is considered as not trustworthy, leaving room for the following (as STRIDE):

- Spoofing:
  - User may attempt to crack or steal another user's password
  - User may attempt to request ticket on behalf of another user

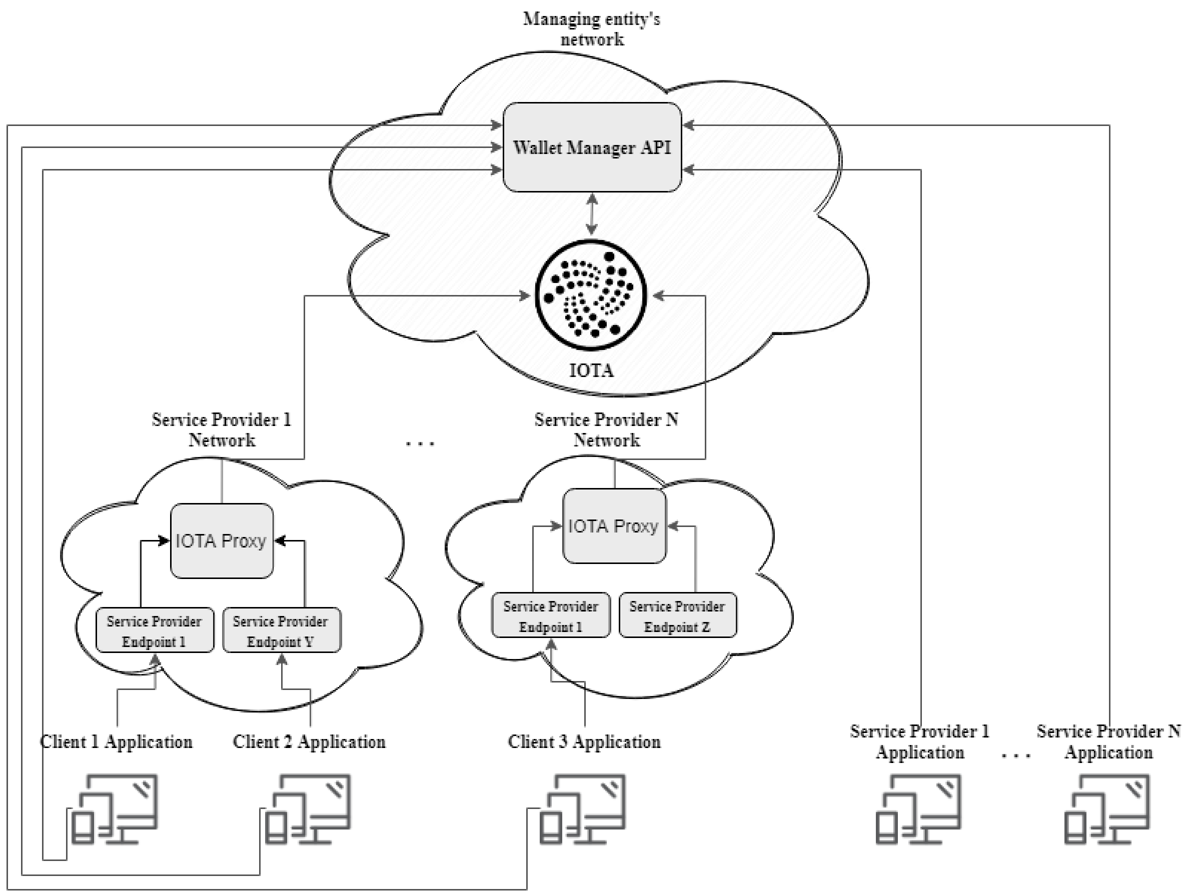

Managing entity's network
Wallet Manager API
IOTA
Service Provider 1 Network
. . .
Service Provider N Network
IOTA Proxy
Service Provider Endpoint 1
Service Provider Endpoint Y
IOTA Proxy
Service Provider Endpoint 1
Service Provider Endpoint Z
Client 1 Application
Client 2 Application
Client 3 Application
Service Provider 1 Application
. . .
Service Provider N Application

Figure 4.7: Execution environment of components of the system

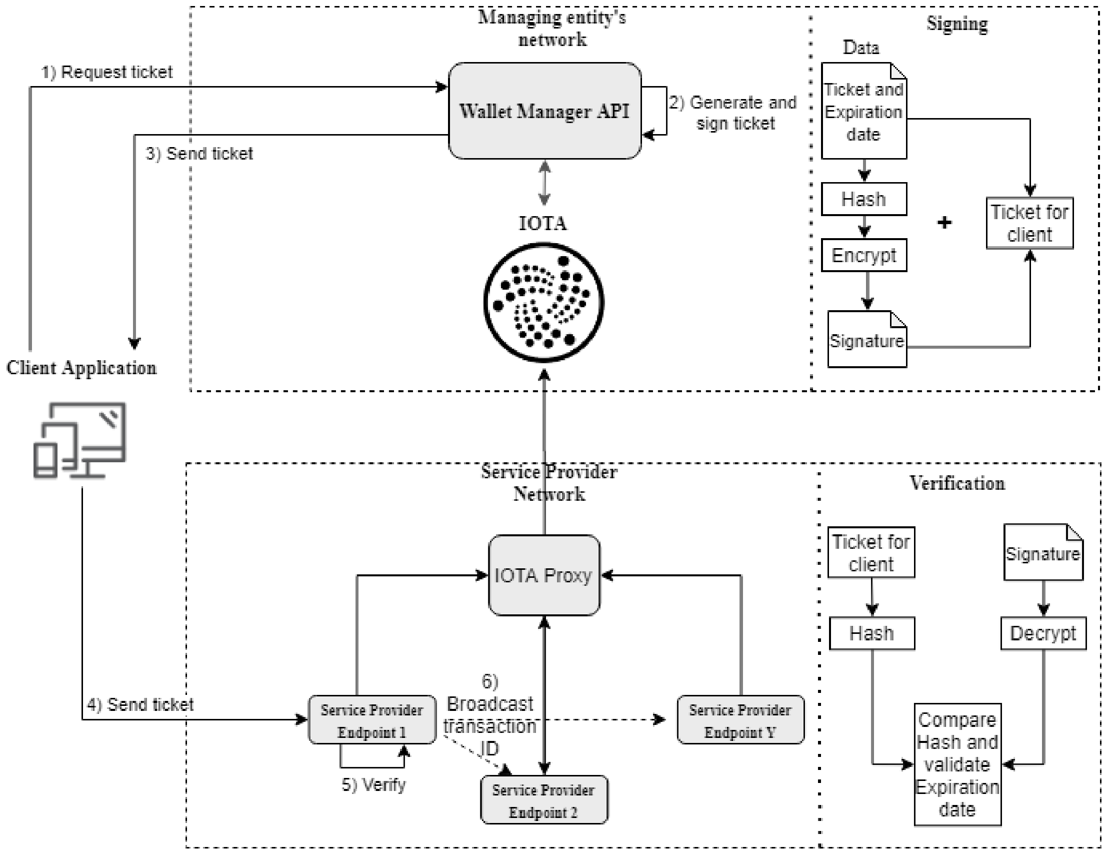
- Tampering:
  - User may attempt to change ticket data before sending to a service provider endpoint
- Repudiation:
  - User can claim he did not sent a ticket
- Information disclosure:
  - User may gather information in the tickets regarding IOTA transactions and addresses
- Denial of service:
  - User can spam requests to flood and degrade the systems
- Elevation of privilege:
  - User may gain access to a service provider account and change the price of the tickets

4.4 Overlayer

This section provides some design decisions that could have been integrated into the implementation of the solution in order to prevent some of the threats and flaws in the IOTA protocol, components or the solution in general.

4.4.1 Security

In this section, a security model which overlays the presented solution is described in order to mitigate some of the threats considered in Section 4.3.2. The threats defined in the categories Spoofing, Denial of service and Elevation of priviledge will not be considered taking into account the trust model defined in Section 4.3.1 - wallet manager and service provider application are trustworthy and secure. The threat Information disclosure will also not be considered since the attacker has no direct access to the network. All considered threats (Tampering and Repudiation) are found in a scenario where a client receives a ticket from the wallet manager and has to broadcast it to the service provider endpoint. To solve them, it is necessary to guarantee authenticity, integrity and non-repudiation of the ticket. These requirements are fulfilled using digital signatures. In order to do so, the wallet manager needs to generate a public-private key pair for each new registered service provider and the public key must be known by all service provider endpoints. After that, the user can then request a ticket for that service provider. The ticket is generated and the aggregation of the ticket with the ticket time of expiration is signed. Both the signature and the ticket are sent to the client. The client then sends this data to the service provider endpoint which will verify the signature and if the ticket did not expire yet. If everything is validated, the transactions is sent to the node and the transaction ID contained in the ticket is broadcasted to all other endpoints in the network. The expiration date together with this broadcast, provide an early attempt to stop double-spending. This process can be also observed in Figure 4.8.



Managing entity's network
Signing
Data
1) Request ticket
Wallet Manager API
2) Generate and sign ticket
3) Send ticket
Ticket and Expiration date
Hash
+
Ticket for client
Encrypt
IOTA
Signature
Client Application
Service Provider Network
Verification
IOTA Proxy
Ticket for client
Signature
Hash
Decrypt
4) Send ticket
Service Provider Endpoint 1
6) Broadcast transaction ID
Service Provider Endpoint Y
5) Verify
Service Provider Endpoint 2
Compare Hash and validate Expiration date

Figure 4.8: Security mechanism diagram

4.4.2 Availability

In terms of availability, the coordinator could run in a cluster with other coordinators. To keep it simple, only one coordinator could be running but they all had a shared database which kept the state of the coordinator (which is by default stored in a local file). Therefore, if the coordinator crashes, others can continue to take the responsibility and create milestones to confirm the transactions. This decentralized design not only removes the single point of failure from the system, but also reduce the risk of centralization.. The wallet manager could also be connected to a load balancer which would distribute its requests through the nodes, making sure that the wallet manager would never be disconnected from the nodes and that no node could be overloaded. This solution can be also observed in Figure 4.9.

4.5 Use Cases

To validate the architecture of this system, four specific use cases were designed to exemplify the main functionalities of the project's features. The check balance and charge wallet use cases are applicable to all transportation models. The use transport use case is applicable in the traditional and taxi models. Finally, the use transport (wallet consumption) use case is applicable to the wallet consumption transportation model. As such, it is possible to cover the usability of the client application in different transportation alternatives in this use cases.

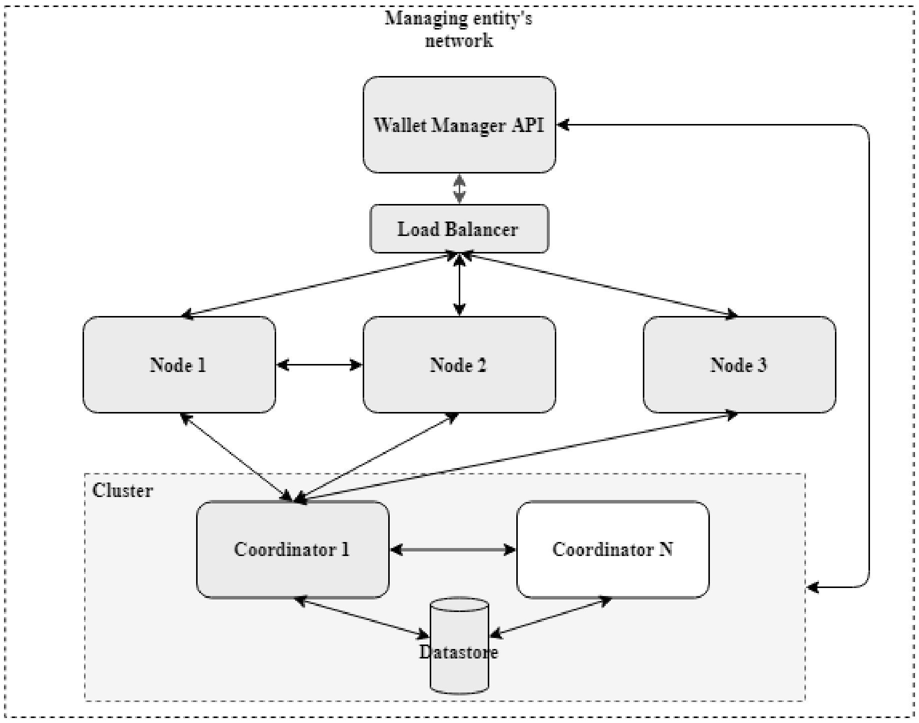


Figure 4.9: Availability mechanism diagram

4.5.1 Check Balance

This operation allows a client or service provider to check its account balance and transaction history. In this use case the client starts by requesting an update of his account status to the wallet manager, which will request that data to a node in the IOTA network. The node responds with the contabilistic balance (obtained from tip transactions - unconfirmed), the confirmed balance (obtained from confirmed transactions), the history of transactions from/to the client and its addresses which contain balance, as depicted in Figure 4.10.

4.5.2 Charge Wallet

This operation transfers balance to the client. In this use case the client starts by requesting a debit to his account to the wallet manager, which will generate a transaction from his address to the client's address. When this process has completed, the wallet manager will confirm the debit to the client, as depicted in Figure 4.11 and Figure 4.11. The necessary steps to generate an IOTA transaction are represented in order to provide a better understanding on further chapters.

4.5.3 Use Transport

This operation allows a client to use a transportation method, transfering value from his account to the service provider. In this use case the client starts by requesting the price of the chosen transport. Then, he requests a ticket to the wallet manager which will generate an unfinalized bundle with more parameters, called a ticket, and send it to the client. The client can then use this ticket by sending it to

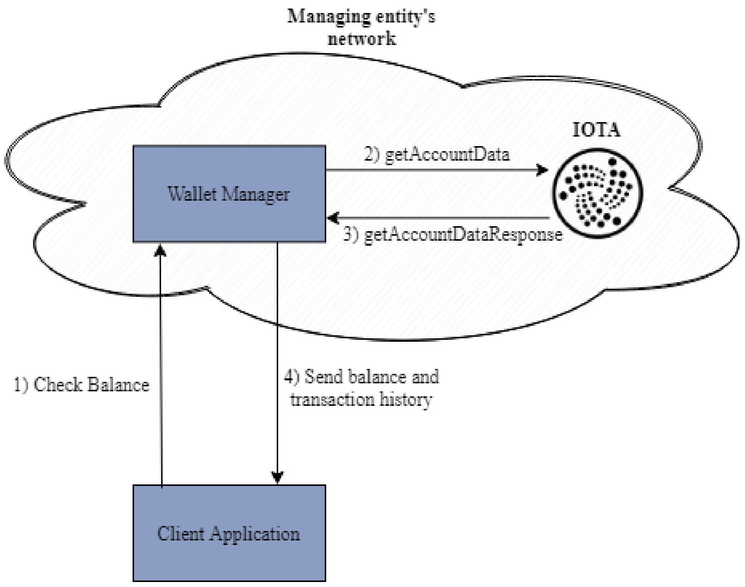

Managing entity's network
IOTA
Wallet Manager
2) getAccountData
3) getAccountDataResponse
1) Check Balance
4) Send balance and transaction history
Client Application

Figure 4.10: Check balance use case diagram

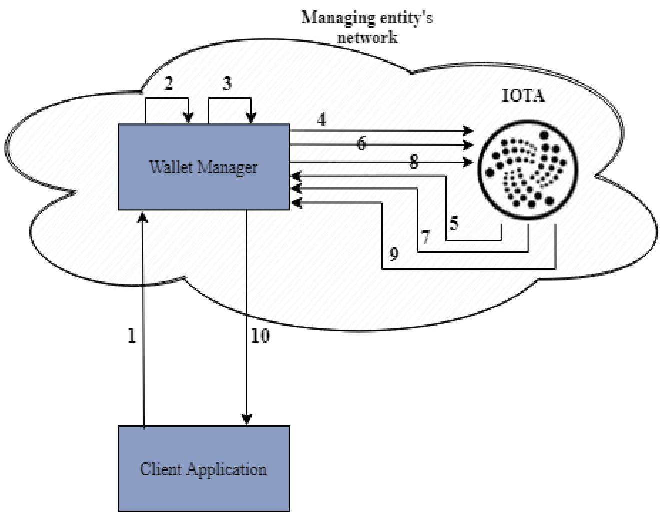


Figure 4.11: Charge wallet use case diagram

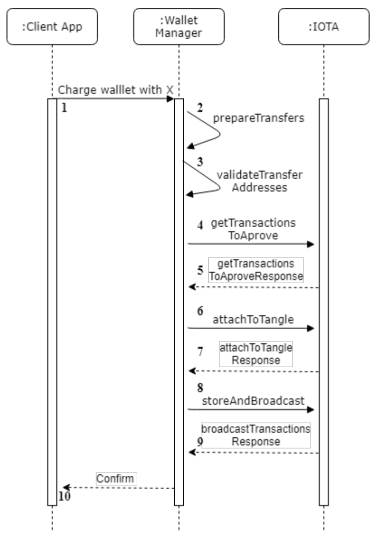


Figure 4.12: Charge wallet sequence diagram

the service provider endpoint. The latter will validate the parameters added by the wallet manager and allow or deny the use of the transport. At the same time, or when available, the proxy will compute the proof of work and broadcast the bundle to a node in the IOTA network, at the service provider endpoint's request. This process is depicted in Figure 4.13 and Figure 4.13. The necessary steps to generate an IOTA transaction are represented in order to provide a better understanding on further chapters.

4.5.4 Use Transport (Wallet Consumption)

This operation allows a client to use a transportation method by doing a check-in and to stop using by doing a check-out. In this use case the client starts by requesting a check-in ticket to the wallet manager which will generate an unfinalized bundle with more parameters and a transactional value of 0, called a check-in ticket, and send it to the client. The client can then use this ticket by sending it to the service provider endpoint. The latter will validate the parameters added by the wallet manager and allow or deny the use of the transport. At the same time, or when available, the proxy will compute the proof of work and broadcast the bundle to a node in the IOTA network, at the service provider endpoint's request. After some time, the client does a check-out, requesting it to the service provider endpoint. The endpoint then calculates the cost of the trip and request a check-out transaction to the wallet manager which will generate an unfinalized bundle with more parameters and a transactional value of the difference between the time of check-out and time of check-in times the service provider rate, called a check-out ticket, and send it to the service provider endpoint. Then the proxy will compute the proof of work and broadcast the bundle to a node in the IOTA network, at the service provider endpoint's request. In the end, the service provider endpoint informs the client of the paid price of the trip. This process is depicted in Figure 4.15 and Figure 4.15. The necessary steps to generate an IOTA transaction are represented in order to provide a better understanding on further chapters.

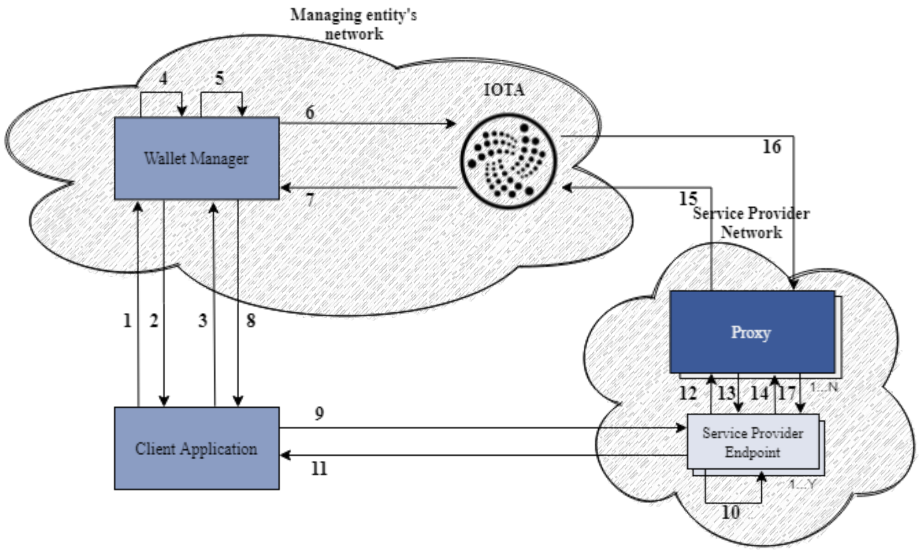


Figure 4.13: Use transport use case diagram

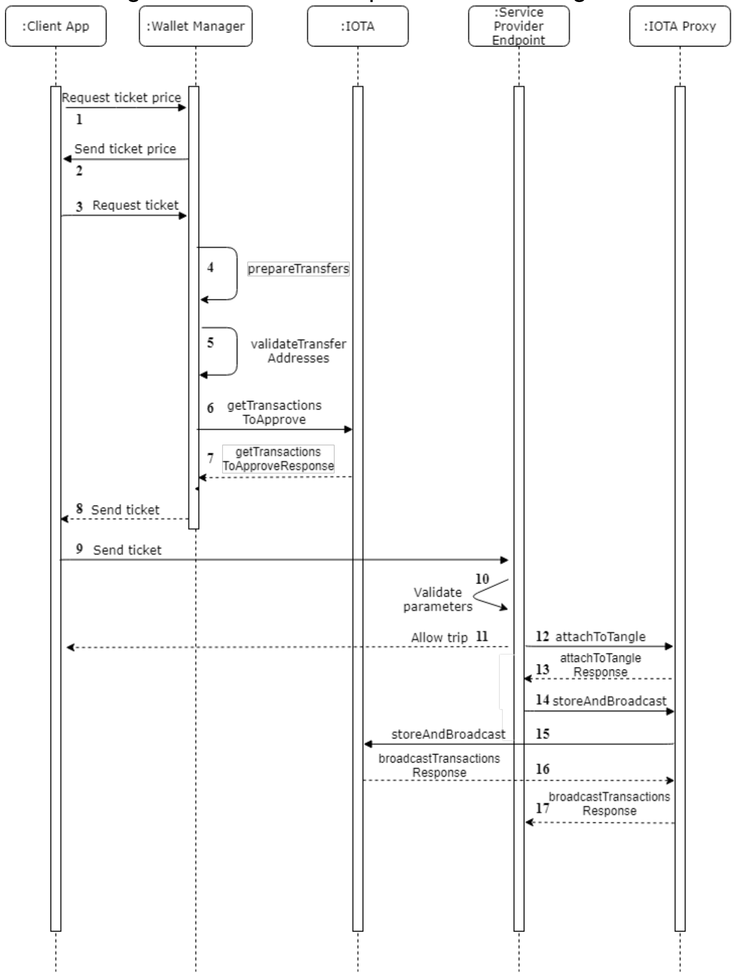


Figure 4.14: Use transport sequence diagram

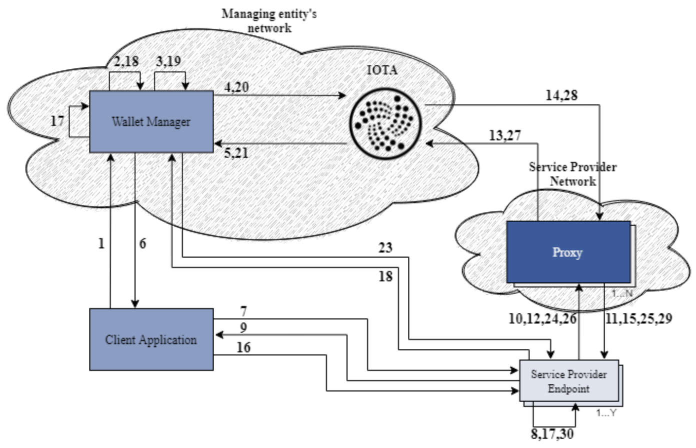
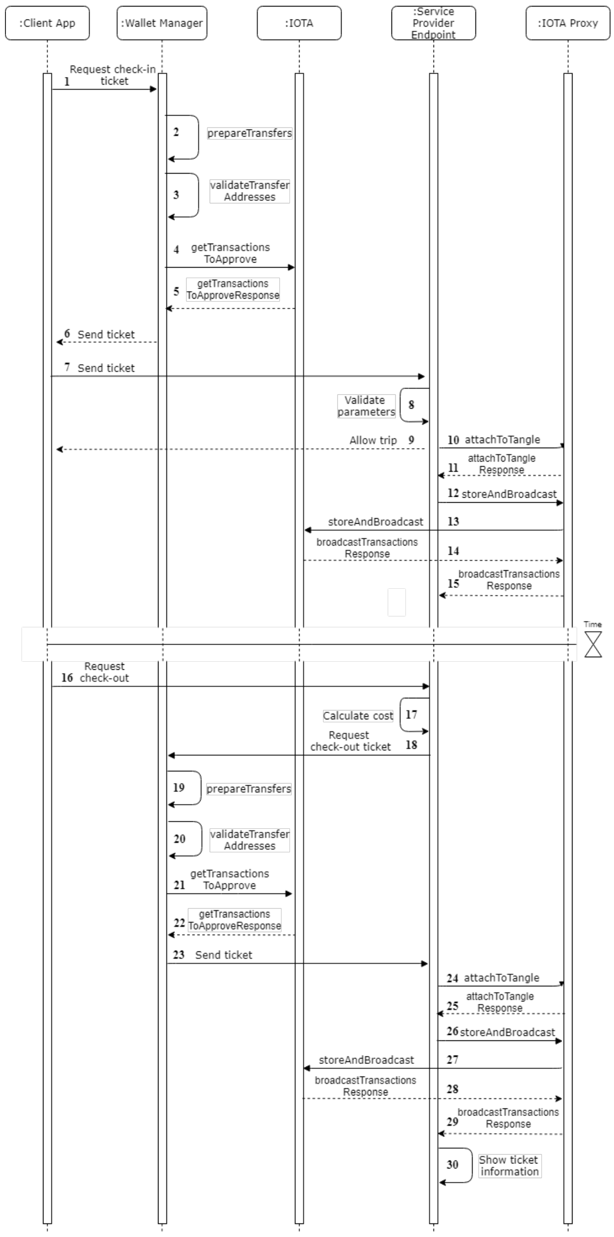


Figure 4.15: Use transport (Wallet Consumption) use case diagram



:Client App
:Wallet Manager
:IOTA
:Service Provider Endpoint
:IOTA Proxy
1 Request check-in ticket
2 prepareTransfers
3 validateTransfer Addresses
4 getTransactions ToApprove
5 getTransactions ToApproveResponse
6 Send ticket
7 Send ticket
8 Validate parameters
9 Allow trip
10 attachToTangle
11 attachToTangle Response
12 storeAndBroadcast
13 storeAndBroadcast
14 broadcastTransactions Response
15 broadcastTransactions Response
Time
16 Request check-out
17 Calculate cost
18 Request check-out ticket
19 prepareTransfers
20 validateTransfer Addresses
21 getTransactions ToApprove
22 getTransactions ToApproveResponse
23 Send ticket
24 attachToTangle
25 attachToTangle Response
26 storeAndBroadcast
27 storeAndBroadcast
28 broadcastTransactions Response
29 broadcastTransactions Response
30 Show ticket information

Figure 4.16: Use transport (Wallet Consumption) sequence diagram

Chapter 5

Implementation

This chapter addresses the implementation details of the developed solution. Section 5.1 presents an overview of all the components of the solution. In the remaining sections each component and operation procedures will be explained in greater detail. The code is also available on Github [1].

5.1 Overview

This section presents an overview of all the components that were implemented, as seen in Figure 5.1. The first implementation step is the creation and configuration of the IOTA network. Once the network is created, it is necessary to launch a program which allows interacting with the network. This program is called wallet manager. Since it is designed to provide operations over the network in a more simplified and permissioned manner to service clients and providers, it was necessary to create an application interface for both. The solution was designed so that applications were to be the only possible triggers of changes in the network.

5.2 IOTA

As presented in Section 4.2.2, the IOTA network can be comprised with many IOTA nodes and an IOTA coordinator. The minimum weight magnitude chosen was 9, as it requires less computing power than 14 (the standard for IOTA as a public network) and is still high enough to demand a high computing power device if the objective is to process a lot of transactions. In combination with the fact that the transactions need to be pre-validated before entering the Tangle, being a permissioned solution, there is no reason to use a proof-of-work effort as high as the IOTA main network. For the sake of simplicity, as many nodes would not provide any advantage or disadvantage compared to a single node in this proof of concept, only one node was implemented. The Figure 5.2 demonstrates the network component's software and communication channels. As opposed to the previously shown diagrams, IOTA proxy is represented in an IOTA network due to it being an IOTA component.

[1] https://github.com/pedro95gomes/79121_dissertation-implementation.git

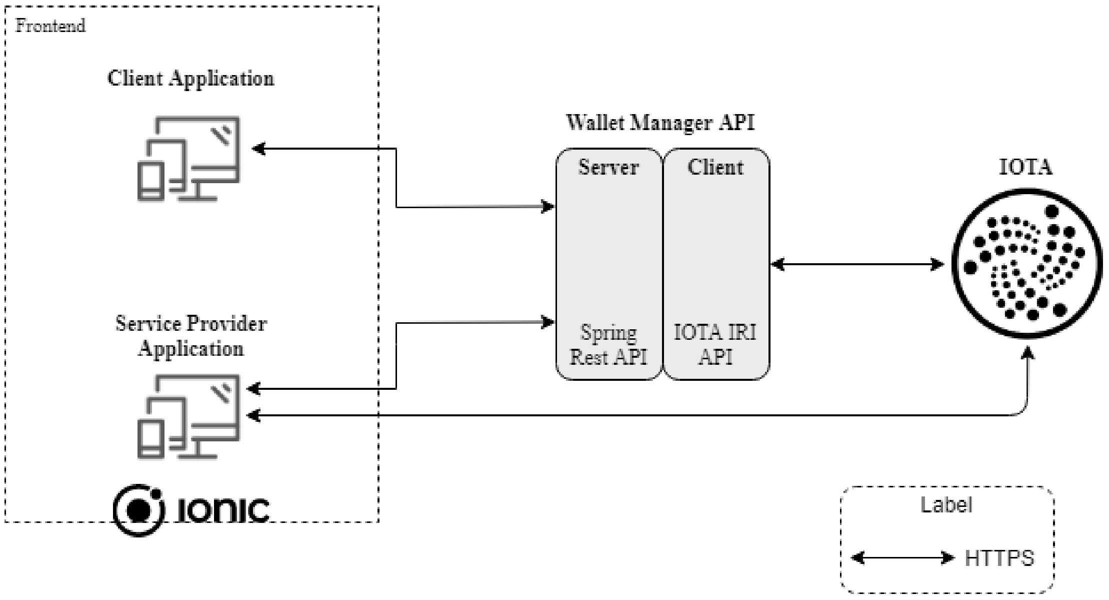


Figure 5.1: Solution's High-Level Implementation Architecture

5.2.1 Node

The node consists in a Ubuntu Server 18.04 with 4GB of RAM and a 64 bit processor, as a minimum requirement recommended by the IOTA organization, and runs the IRI software in a Docker [53] container. IRI [54] is open-source Java software for the IOTA protocol. It connects to a local RocksDB [55] to store ledger data, such as transactions, transaction-metadata, milestones, addresses, snapshots, and others. Each node has its own copy of the ledger and in order to be synchronized with the network, nodes use a gossip protocol to communicate with their neighbors. This communication is done via socket port 15600 using TCP as the transport layer protocol. On the other hand, the IRI API is available on the socket port port 14265 using TCP as the transport layer protocol. As such, both ports need to be open. All this communications are wrapped using HTTPS as the application layer protocol. The host needs to be labeled with either a static IP address or connected to a dynamic DNS service.

5.2.2 Proxy

The proxy has no recommended hardware prerequisites. As such, it also consists in a Ubuntu Server 18.04 with 4GB of RAM and a 64 bit processor and runs an implementation of Caddy [56] that uses IOTA middleware. This middleware allows the server to intercept attachToTangle calls to a node's endpoint and do the proof-of-work. All other requests are forwarded to the IOTA node. It does not need a database since it does not store any data. The communication with the node is done via port 14265 using TCP as the transport layer protocol and HTTPS as the application layer protocol. The host needs to be labeled with either a static IP address or connected to a dynamic DNS service. All connections are made to the proxy through the port 15265.

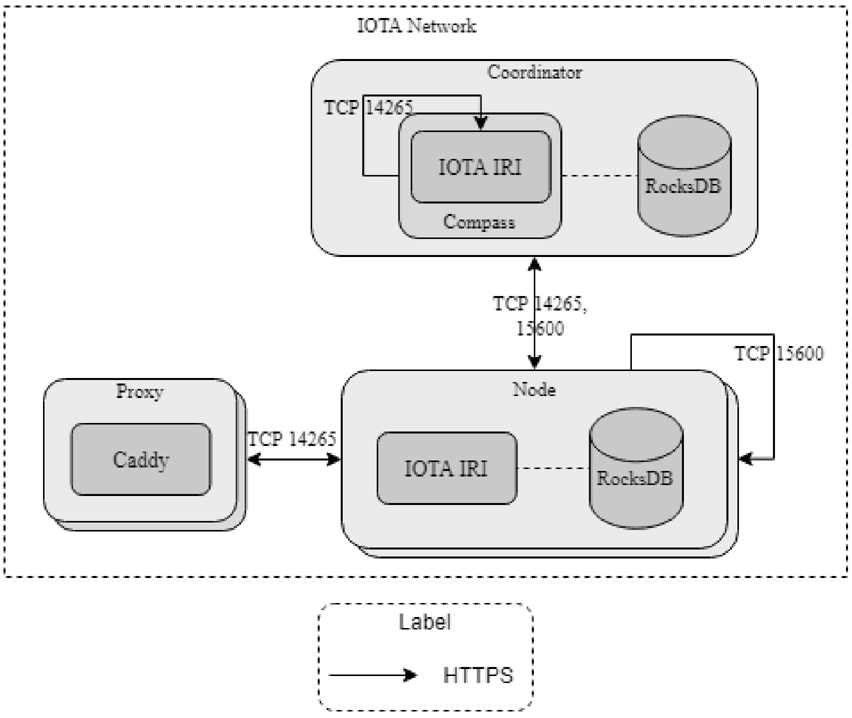


Figure 5.2: IOTA Network Detailed Architecture

5.2.3 Coordinator

The coordinator also consists in a Ubuntu Server 18.04 with 4GB of RAM and a 64 bit processor, as a minimum requirement recommended by the IOTA organization, and runs both the Compass and IRI software in a Docker container. Compass [57] is an open-source Java implementation of an IOTA Network coordinator. The compass software issues milestones, which are transactions signed by the coordinator, to the local IRI. Since milestones behave like transactions, the behavior and configuration of a coordinator is very similar to the node. It connects to a local RocksDB to store ledger data, such as transactions, transaction-metadata, milestones, snapshots, addresses, and others. The coordinator has its own copy of the ledger and in order to be synchronized with the network, it uses a gossip protocol to communicate with their neighbors. This communication is done via socket port 15600 using TCP as the transport layer protocol. On the other hand, the IRI API is available on the socket port port 14265 using TCP as the transport layer protocol. As such, both ports need to be open. All this communications are wrapped using HTTPS as the application layer protocol. The host needs to be labeled with either a static IP address or connected to a dynamic DNS service.

5.2.4 Setup

To use the a private IOTA network, a coordinator, a proxy and a node were created. These were setup in a virtualized environment (VMware [58]). The most relevant configuration parameters for the IRI, Caddy and Compass software can be found in Tables 5.1, 5.2 and 5.3 respectively. After configuring each component, the first node on the network must be launched from a snapshot containing the genesis transaction of the network, that is, the starting state of the network. In the initial state of this solution's

Parameter	Value	Note
Minimum weight magnitude	9	Minimum accepted proof-of-work effort
Neighbors	protocol://ip-address:15600	List of node neighbors
Testnet	True	Run node on a network other than the Mainnet
Testnet Coordinator	coordinator-address	81-trytes address of the testnet Coordinator

Table 5.1: IRI configuration parameters

Parameter	Value	Note
Maximum weight magnitude	9	Maximum accepted proof-of-work effort
Node	protocol://ip-address:14265	Node to which the proxy is connected to
Maximum transactions per call	10000	

Table 5.2: Caddy configuration parameters

network, all available IOTA tokens are associated to a wallet manager's address. Therefore, it is necessary to create a seed and generate an address for the entity that manages this solution beforehand. All the IOTA tokens in circulation on the network will be distributed by this seed. After every component is running, the network is ready.

5.3 Wallet Manager API

The wallet manager is a REST API developed using the Spring Boot [59] for Java. This component can be seen as both a server and a client. A server that receives requests to its API and a client that sends requests to nodes using the IOTA IRI API. A simple database was created in order to store the clients and service providers seeds, ticket costs and rates, service provider's addresses and login information. This database was implemented in SQLite [60] as it simple to use and fulfilled the demands of this proof-of-concept.

5.3.1 Server (Spring REST API)

In order to differentiate the APIs for the different types of functionalities, two controllers were created. One controller is the ClientController, which purpose is to serve the client requests. Its operations are described in Table 5.4 and documented in Figure 5.3. The second is the ServiceProviderController and it serves the service provider requests. Its operations are described in Table 5.5 and documented in Figure 5.3. This documentation was produced using Spring's MockMvc [61] with JUnit [62] tests. The

Parameter	Value	Note
Minimum weight magnitude	9	Minimum accepted proof-of-work effort
Security Level	1	Security level of Compass' private key/address pairs
Tick	60	Number of milliseconds Compass waits after creating a milestone
Host	localhost	URL of the IOTA node to which Compass sends milestones
Depth	14	Depth of the merkle tree

Table 5.3: Compass configuration parameters

Operation	**Action**
Create User	Registers user login information in database
Login	Login client
Check Balance	Get user's balance, transaction history and addresses with tokens
Charge Wallet	Transfers tokens to a client address
Transport Price	Get requested service provider's transport price
Taxi Price	Get requested service provider's transport price based on distance
Ticket	Generates a ticket for the selected service provider transportation option
Check-in ticket	Generates a check-in ticket for the selected service provider transport
Check-out ticket	Generates a check-out ticket for the check-in ticket previously generated

Table 5.4: ClientController description of operations

Operation	**Action**
Create Service Provider	Registers service provider login information in database
Login	Login service provider
Check Balance	Get providers's balance, transaction history and addresses with tokens
Updated Price	Change service provider transport price
Update Address	Generate a new receiver address for the service provider

Table 5.5: ServiceProviderController description of operations

response body of some operations had to be trimmed due to its size.

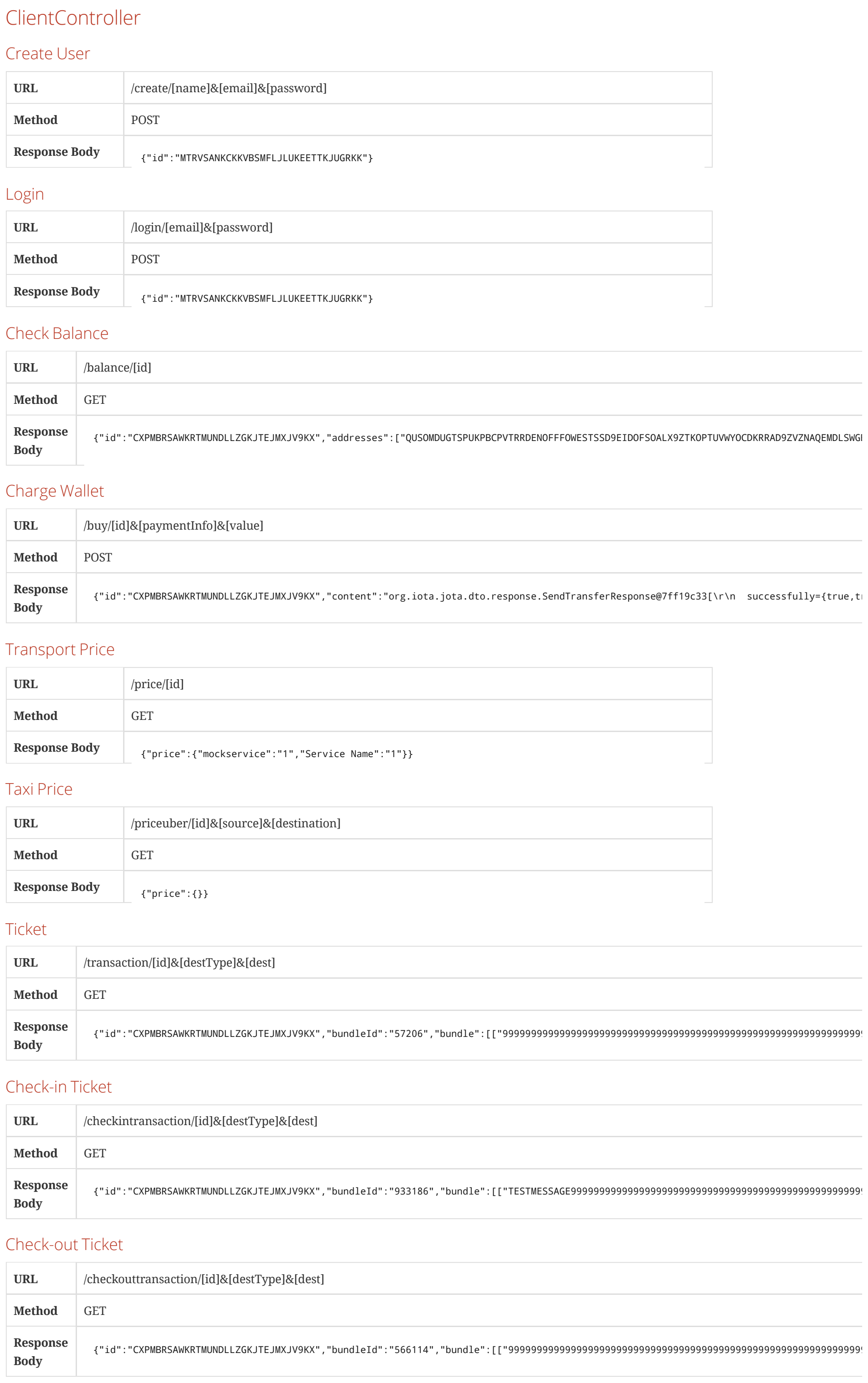
5.3.2 Client (IOTA IRI API)

The client was developed using Jota (Official IOTA client library for Java) [63] which implements the IOTA IRI API. All API calls are made synchronously since the library is in beta and does not support asynchronous requests yet. This does not affect the solution as a proof-of-concept, however it is essential for a large scale deployment. The core operations performed when sending a transaction are prepareTransfers, validateTransferAddresses, attachToTangle, storeTransactions and broadcastTransactions, as refered in previously shown diagrams. All used Jota operations are described in Table 5.6

5.4 Frontend

The frontend consists in a single multi-platform application whose intent is to provide an interface to use the Wallet Manager API. It was developed using the Ionic framework. Ionic [64] is an open-source SDK that provides tools and services for developing hybrid mobile, desktop, and web applications using web technologies such as Angular [65], React [66] or Vue.js [67]. In order to use the application, both the client and the service provider need be registered and logged in. When logged in, the wallet manager provides the user's ID which will be the identifier for conducting all operations and these will be redirected to the client application or service provider applications respectively. The client application operations are described in Table 5.7 and the service provider application operations in Table 5.8.

The service provider endpoint simulates a transport gateway/terminal/device behavior and was also implemented in the service provider application. The other option was to implement it on a RaspberryPi [68], however there were no advantages in doing it besides being more truthful to real scenarios, like the



ClientController

Create User

URL	/create/[name]&[email]&[password]
Method	POST
Response Body	`{"id":"MTRVSANKCKKVBSMFLJLUKEETTKJUGRKK"}`

Login

URL	/login/[email]&[password]
Method	POST
Response Body	`{"id":"MTRVSANKCKKVBSMFLJLUKEETTKJUGRKK"}`

Check Balance

URL	/balance/[id]
Method	GET
Response Body	`{"id":"CXPMBRSAWKRTMUNDLLZGKJTEJMXJV9KX","addresses":["QUSOMDUGTSPUKPBCPVTRRDENOFFFOWESTSSD9EIDOFSOALX9ZTKOPTUVWYOCDKRRAD9ZVZNAQEMDLSWG`

Charge Wallet

URL	/buy/[id]&[paymentInfo]&[value]
Method	POST
Response Body	`{"id":"CXPMBRSAWKRTMUNDLLZGKJTEJMXJV9KX","content":"org.iota.jota.dto.response.SendTransferResponse@7ff19c33[\r\n  successfully={true,t`

Transport Price

URL	/price/[id]
Method	GET
Response Body	`{"price":{"mockservice":"1","Service Name":"1"}}`

Taxi Price

URL	/priceuber/[id]&[source]&[destination]
Method	GET
Response Body	`{"price":{}}`

Ticket

URL	/transaction/[id]&[destType]&[dest]
Method	GET
Response Body	`{"id":"CXPMBRSAWKRTMUNDLLZGKJTEJMXJV9KX","bundleId":"57206","bundle":[["999999999999999999999999999999999999999999999999999999999999999`

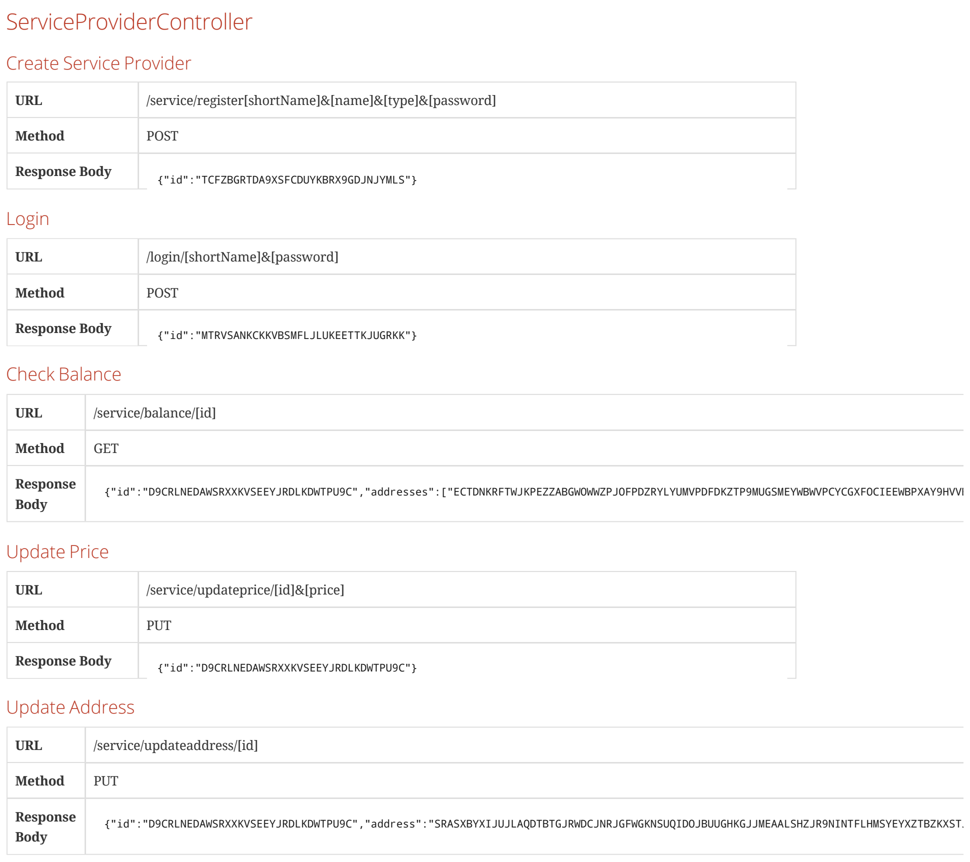
Check-in Ticket

URL	/checkintransaction/[id]&[destType]&[dest]
Method	GET
Response Body	`{"id":"CXPMBRSAWKRTMUNDLLZGKJTEJMXJV9KX","bundleId":"933186","bundle":[["TESTMESSAGE9999999999999999999999999999999999999999999999999`

Check-out Ticket

URL	/checkouttransaction/[id]&[destType]&[dest]
Method	GET
Response Body	`{"id":"CXPMBRSAWKRTMUNDLLZGKJTEJMXJV9KX","bundleId":"566114","bundle":[["99999999999999999999999999999999999999999999999999999999999999`

Figure 5.3: ClientController endpoint operations


ServiceProviderController

Create Service Provider

URL	/service/register[shortName]&[name]&[type]&[password]
Method	POST
Response Body	`{"id":"TCFZBGRTDA9XSFCDUYKBRX9GDJNJYMLS"}`

Login

URL	/login/[shortName]&[password]
Method	POST
Response Body	`{"id":"MTRVSANKCKKVBSMFLJLUKEETTKJUGRKK"}`

Check Balance

URL	/service/balance/[id]
Method	GET
Response Body	`{"id":"D9CRLNEDAWSRXXKVSEEYJRDLKDWTPU9C","addresses":["ECTDNKRFTWJKPEZZABGWOWWZPJOFPDZRYLYUMVPDFDKZTP9MUGSMEYWBWVPCYCGXFOCIEEWBPXAY9HVV`

Update Price

URL	/service/updateprice/[id]&[price]
Method	PUT
Response Body	`{"id":"D9CRLNEDAWSRXXKVSEEYJRDLKDWTPU9C"}`

Update Address

URL	/service/updateaddress/[id]
Method	PUT
Response Body	`{"id":"D9CRLNEDAWSRXXKVSEEYJRDLKDWTPU9C","address":"SRASXBYXIJUJLAQDTBTGJRWDCJNRJGFWGKNSUQIDOJBUUGHKGJJMEAALSHZJR9NINTFLHMSYEYXZTBZKXST`

Figure 5.4: ServiceProviderController endpoint operations

Operation	Action
attachToTangle	Does proof of work for the given transaction
broadcastTransactions	Sends transaction trytes to a node
checkConsistency	Checks the consistency of transactions
findTransactions	Finds transactions that contain the given values in their fields
generateNewAddresses	Checks if a set of transactions is confirmed
getAccountData	Returns addresses balance, transaction history and associated tokens
getBalances	Calculates the confirmed and unconfirmed balance of addresses
getInclusionStates	Checks if a set of transactions is confirmed
getInputs	Gets the input addresses of a seed
getTransactionsToApprove	Gets two consistent tip transaction hashes to use as tips
getTransfers	Finds all the bundles for all the addresses based on the seed
getTrytes	Gets a transaction's contents in trytes
prepareTransfers	Prepares transfer by generating bundle, finding and signing inputs
sendTransfer	Runs prepareTransfers, attachToTangle and then storeAndBroadcast
storeAndBroadcast	Runs storeTransactions and broadcastTransactions
storeTransactions	Stores transactions in a node's view of the Tangle
validateTransfersAddresses	Validates the supplied transactions for correct input/output and key reuse
wereAddressSpentFrom	Check if a list of addresses was ever spent from

Table 5.6: Jota library used operations

Operation	Wallet Manager API endpoints	Action
Update Balance	GET /balance	Balance and Transaction History Update
Buy Tokens	POST /buy	Token acquisition
Tickets	[GET /price or GET /uberprice and GET /transaction] or GET /checkintransaction	Ticket generation

Table 5.7: Client application operations

Metro or Bus, and it would complicate the simulation. As such, whenever a client has generated a ticket, he will submit it on the service provider application. After that, the endpoint will validate its parameters, retrieve the bundle in the ticket, ask a node/proxy to compute the proof-of-work for all transactions in the bundle and then send it. The service provider endpoint operations can be found in Table 5.9.

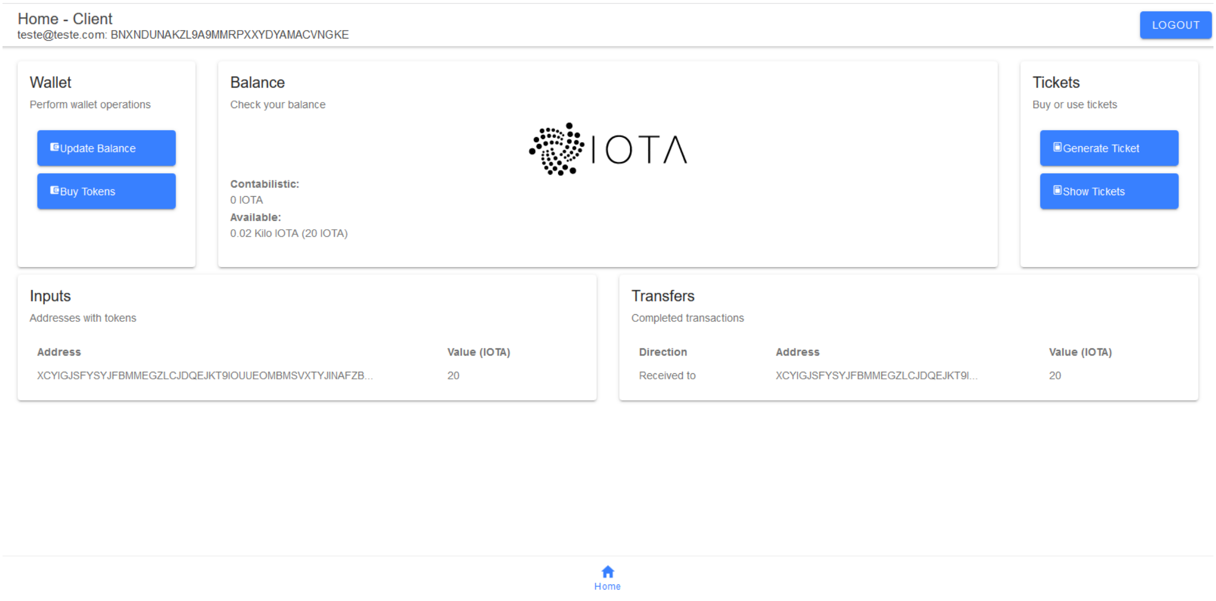
The web interface of the client application can be seen in Figure 5.5 and the service provider application in Figure 5.6.

Operation	Wallet Manager API endpoints	Action
Update Balance	GET /service/balance	Balance and Transaction History Update
Update Price	PUT /service/updateprice	Update service provider's transport price
Update Address	PUT /service/updateaddress	Generate new receiver address

Table 5.8: Service provider application operations

Operation	**Wallet Manager API endpoints**	**Action**
Receive Ticket	None	Receive ticket from client and send to node
Checkout Ticket	GET /checkouttransaction	Request check-out ticket and then send it to node

Table 5.9: Service provider endpoint operations


Home - Client
teste@teste.com: BNXNDUNAKZL9A9MMRPXXYDYAMACVNGKE

Wallet
Perform wallet operations

Balance
Check your balance

Contabilistic:
0 IOTA
Available:
0.02 Kilo IOTA (20 IOTA)

Tickets
Buy or use tickets

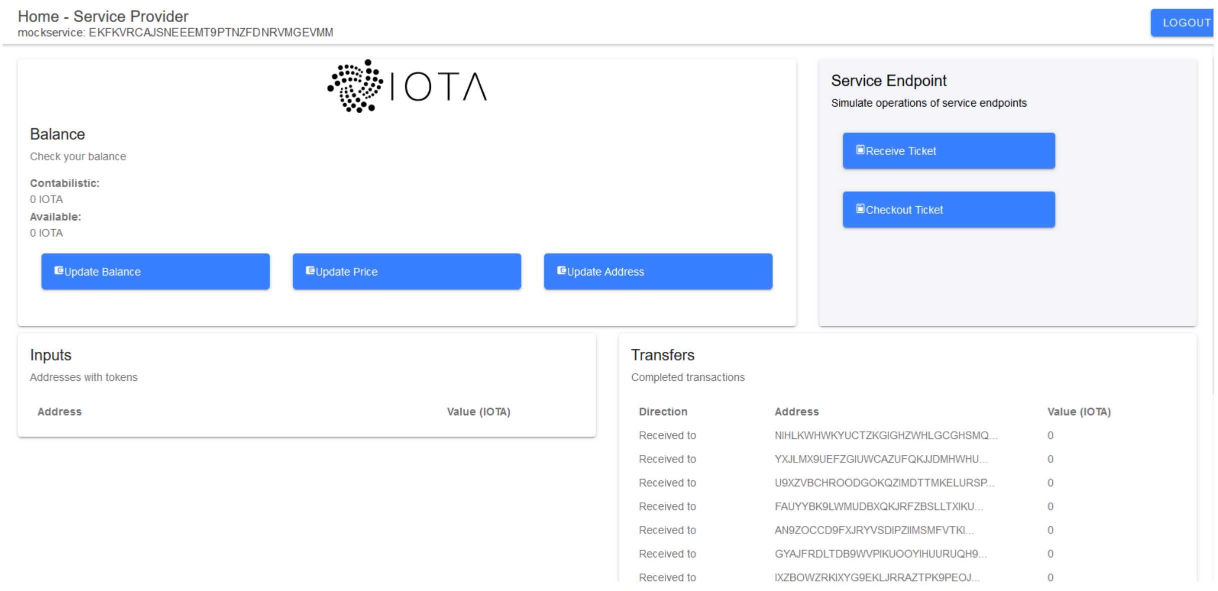
Inputs
Addresses with tokens

Address	Value (IOTA)
XCYIGJSFYSYJFBMMEGZLCJDQEJKT9IOUUEOMBMSVXTYJINAFZB...	20

Transfers
Completed transactions

Direction	Address	Value (IOTA)
Received to	XCYIGJSFYSYJFBMMEGZLCJDQEJKT9I...	20

Figure 5.5: Client application interface - After buying tokens which were not yet confirmed


Home - Service Provider
mockservice: EKFKVRCAJSNEEEMT9PTNZFDNRVMGEVMM

Balance
Check your balance

Contabilistic:
0 IOTA
Available:
0 IOTA

Service Endpoint
Simulate operations of service endpoints

Inputs
Addresses with tokens

Address	Value (IOTA)

Transfers
Completed transactions

Direction	Address	Value (IOTA)
Received to	NIHLKWHWKYUCTZKGIGHZWHLGCGHSMQ...	0
Received to	YXJLMX9UEFZGIUWCAZUFQKJJDMHWHU...	0
Received to	U9XZVBCHROODGOKQZIMDTTMKELURSP...	0
Received to	FAUYYBK9LWMUDBXQKJRFZBSLLTXIKU...	0
Received to	AN9ZOCCD9FXJRYVSDIPZIIMSMFVTKI...	0
Received to	GYAJFRDLTDB9WVPIKUOOYIHUURUQH9...	0
Received to	IXZBOWZRKIXYG9EKLJRRAZTPK9PEOJ...	0

Figure 5.6: Service provider application interface - After receiving multiple zero-valued transactions

Chapter 6

Results

This chapter presents the experimental evaluation of IOTA network, in Section 6.1, with the objective of extending previous benchmarks performed on IOTA with more metrics, which were vital to designing the solution. Functional tests were also performed on the designed solution, as seen in Section 6.2. Finally, a discussion of the obtained results is presented in Section 6.3.

6.1 Performance Tests

The state-of-the-art documents regarding IOTA performance and throughput provide some sound conclusions that help designing an IOTA network for a practical application, however they are not enough. Some of these results state that: (1) Currently battery-powered IoT devices are not suitable for IOTA operations, (2) Transaction speed has a good linear scalability against the number of senders, (3) Confirmed transactions per second grow almost linearly with Transaction arrival rate and (4) Proof-of-work (attachToTangle) and tip selection (getTransactionsToApprove) are the operations that contribute more over-all transaction insertion delay of each transaction. The design of the network was the base for all the other solution design choices and, for this, it was required to answer some questions: (1) Which type of device will compute the proof-of-work, (2) How does different proof-of-work efforts affect its performance, (3) What are this device's hardware requirements, (4) How many nodes/proxies will the network require and (5) Can the network withstand a large amount of transactions / How does it react when overcharged. The next sections will try to answer this questions.

6.1.1 Platform and Evaluation Metrics

The benchmark consists in generating and sending multiple transactions at the same time to a node or/and a proxy. The process of generating and sending a single transaction is represented in Figure 6.1 for a node and in Figure 6.2 for a proxy. IOTA nodes and proxies will be evaluated on two different scenarios via the following metrics: (1) Tip selection (getTips) and proof-of-work (attachToTangle) computation times, (2) Transactions completed per second (Throughput), (3) Time to generate/complete transaction (Latency), (4) Time to confirm transactions (Confirm Latency), (5) CPU and Memory usage.

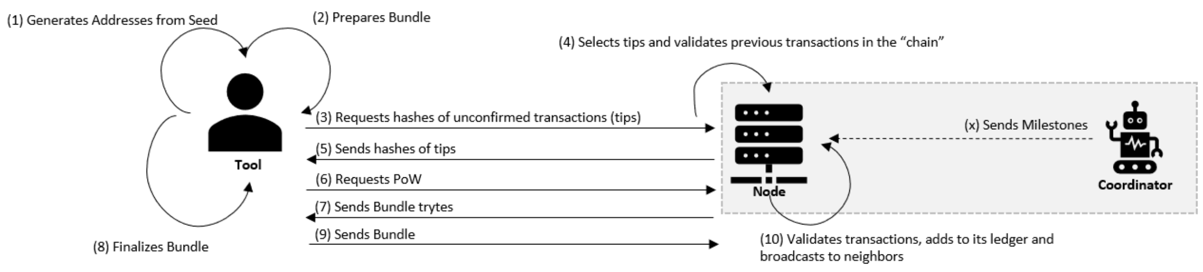

(1) Generates Addresses from Seed
(2) Prepares Bundle
(4) Selects tips and validates previous transactions in the "chain"
Tool
(3) Requests hashes of unconfirmed transactions (tips)
(5) Sends hashes of tips
(6) Requests PoW
(7) Sends Bundle trytes
(9) Sends Bundle
(8) Finalizes Bundle
Node
(x) Sends Milestones
Coordinator
(10) Validates transactions, adds to its ledger and broadcasts to neighbors

Figure 6.1: Generating and sending a transaction to a node

On both scenarios there were also some network variants such as: (1) Type of machine (node or proxy), (2) Number of virtual CPU's/logical cores in the machine (ranging from 1, 2 or 4), (3) Proof-of-work effort (Minimum weight magnitude of 3, 6, 9, 11 or 13) and (4) Different workloads (Number of sent transactions). To achieve this, a self-made benchmark tool was created using Python as the coding language and Pyota [69] as the official Python client library for IOTA since it was the only library that supported asynchronous requests to the nodes. Python, however, does not provide the functionality of "true" threads due to GIL. GIL (Python Global Interpreter Lock) [70] is a mutex (or a lock) that allows only one thread to hold the control of the Python interpreter. This means that only one thread can be in a state of execution at any point in time. Despite this, since most of the tasks were network I/O bound achieving concurrency with asynchronous IO was enough for the task. It was implemented using the asyncio library [71]. Then, after verifying that the tool was not using all the available computing power of the machine that ran the benchmarking tool, the multiprocessing library [72] was used in order to take full advantage of every logical core. On the other hand, the IOTA node was configured to be the publisher of ZeroMQ [73] events, regarding transactions that the IRI node has recently appended to the ledger and transactions that had recently been confirmed, and the tool acted as subscriber. The computational resources of the hosts were logged using the top [74] tool in Linux at a rate of one measure per second. All this data was then compiled to a database and analyzed using Power BI [75]. The environment in which the benchmark was performed consists in two physical machines connected in a LAN. One ran a node and a proxy in a virtualized environment with 4GB of RAM dedicated to each and the number of logical processors of the virtual machines varied between 1, 2 and 4. This physical machine has the following hardware specifications: an AMD Ryzen 7 3700x 8-Core 3.6Ghz with 16 logical processors and 32GB of RAM. The other machine ran the developed benchmark tool once per core variation, that is three runs for the node and three runs for the proxy, making a total of 6 runs. Its hardware specifications are: an Intel I5-8300H 4-Core 2.3Ghz with 8 logical processors and 16GB of RAM. The next sections will describe the gathered results obtained from this tool in different scenarios.

6.1.2 Experiment Results

Scenario I

In Scenario I, all previously described network variants were used. Analysing the previously defined metrics, the following observations were produced:

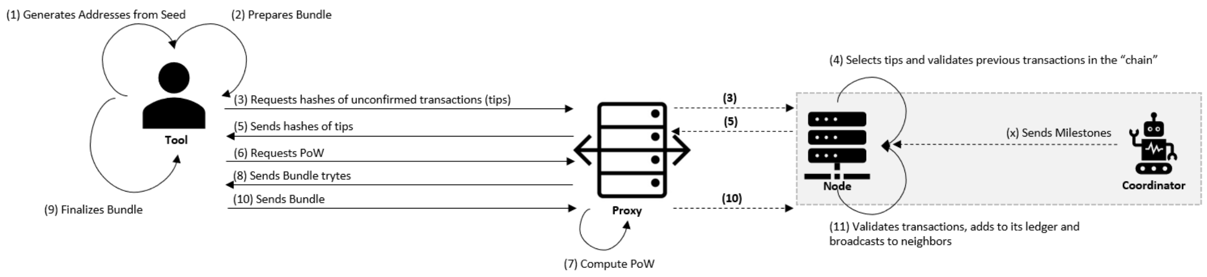
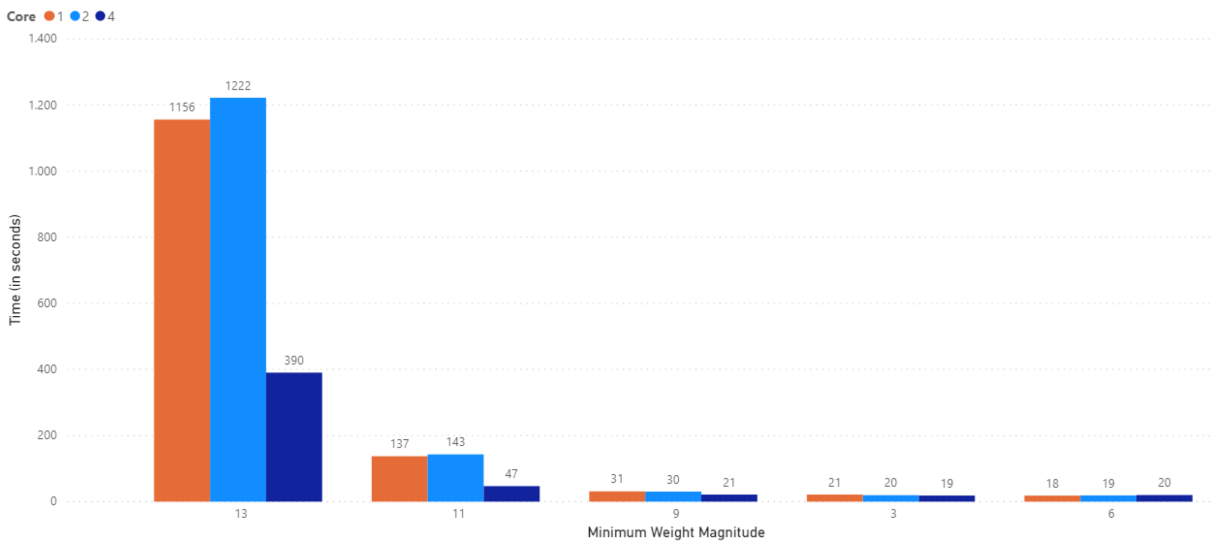
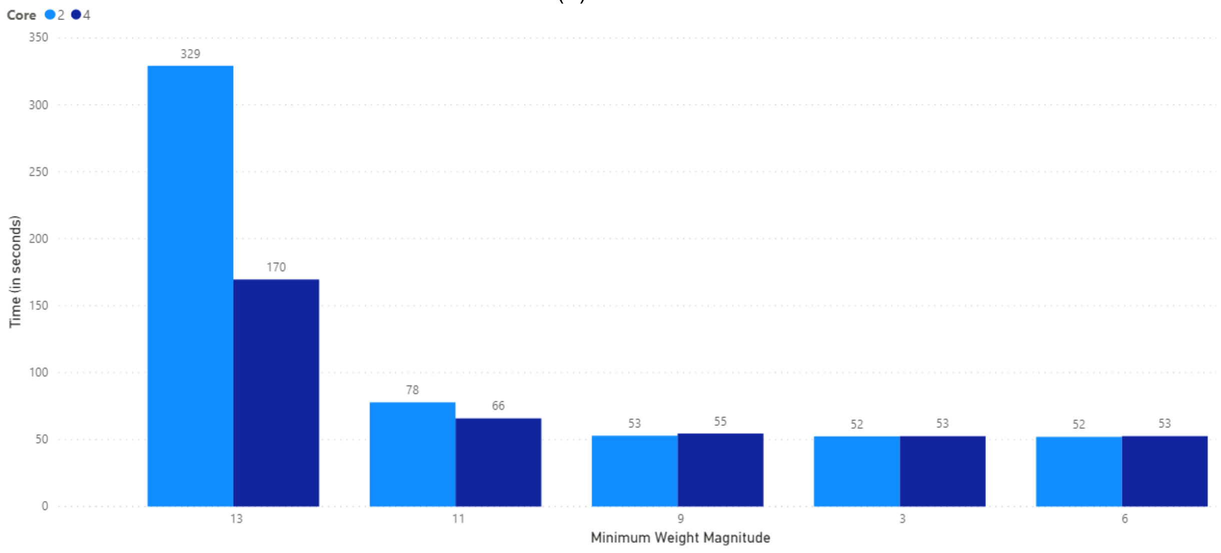

(1) Generates Addresses from Seed
(2) Prepares Bundle
Tool
(9) Finalizes Bundle
(3) Requests hashes of unconfirmed transactions (tips)
(5) Sends hashes of tips
(6) Requests PoW
(8) Sends Bundle trytes
(10) Sends Bundle
Proxy
(7) Compute PoW
(3)
(5)
(10)
(4) Selects tips and validates previous transactions in the "chain"
Node
(x) Sends Milestones
Coordinator
(11) Validates transactions, adds to its ledger and broadcasts to neighbors

Figure 6.2: Generating and sending a transaction to a proxy

1. Proof-of-work (attachToTangle) and Tip selection (getTransactionsToApprove) computation times: From the data gathered, in Figure 6.3, it is possible to observe how the computation times of the attachToTangle operation change for different minimum weight magnitude values and number of logical cores in different machines (the node and proxy). This data was an averaged for different numbers of sent transactions. What is important to note here is how the computation times of this operation remain virtually the same until a certain threshold weight for all machines and number logical cores associated. This threshold weight can be seen at around the minimum weight magnitude of 9. Of course the number of sent transactions also plays a critical role in increasing or decreasing the computations times, as more sent transactions will require the same computational resources to process more transactions, thus distributing the computer power between more operations. On low numbers of sent transactions, some of the computational times are equivalent regarding the minimum weight magnitude, like the minimum weight magnitude of 11 on a node, or of 13 on a proxy. Despite all this, it is clear how both the node and proxy require more logical cores on high demanding tasks to be able to compute attachToTangle in reasonable times. It is also clear that the proxy is able to handle high demanding tasks better than the node, computing the attachToTangle operation in less time than the node with half the cores. To resume, all of these variables seem to affect the performance of the machines when computing the attachToTangle operation. From the data gathered, in Figure 6.4, it is possible to observe how the computation times of the getTransactionToApprove operation change for different number of sent transactions and number of logical cores in different machines (the node and proxy). It is important to reinforce, at this stage, that this operation is not actually executed in the proxy but in the node, so when referring to the proxy it's implicit that is actually the performance of a node with a proxy relaying the requests for this operations. As such, it is possible to compare the performance of a node in a scenario where it is also computing the proof-of-work to a scenario where it does not and therefore has more computational resources. In both Figure 6.4 and 6.4, the computational times increase as more transactions are sent (for all logical cores). However, it seems to reach an upper bound limit between 30 and 40 seconds after a certain number of sent transactions. It's also visible that increasing the number of cores have little to no effect on the performance of this operation. There is, however, another parameter that influences its performance: the milestone depth. The depth defines how many milestones in the past the node starts the tip selection algorithm. The greater the depth, the farther back in the Tangle the IOTA node starts. A greater depth increases the time



(a) Node

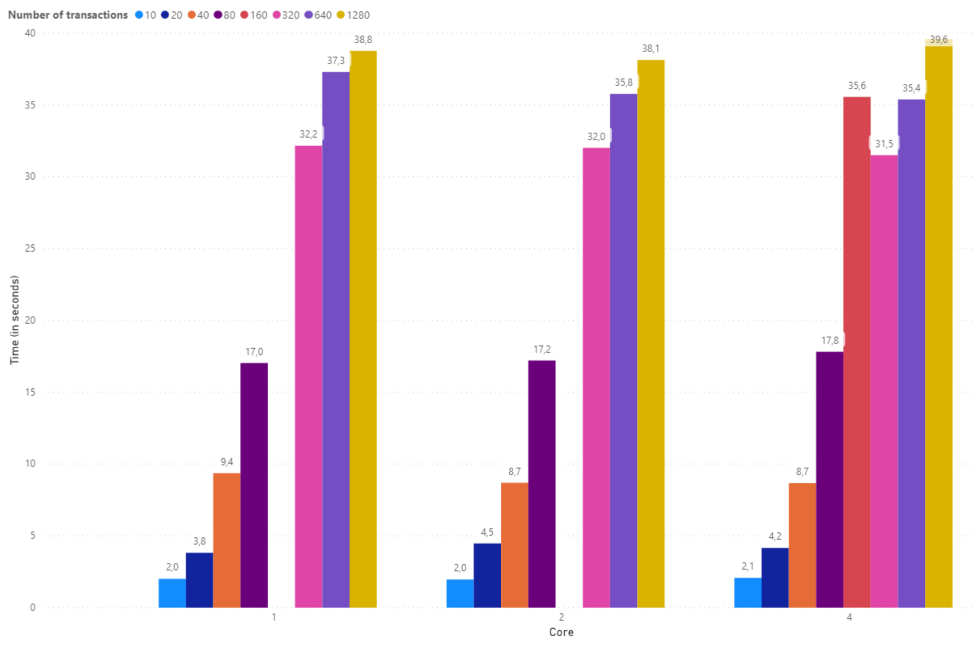


(b) Proxy

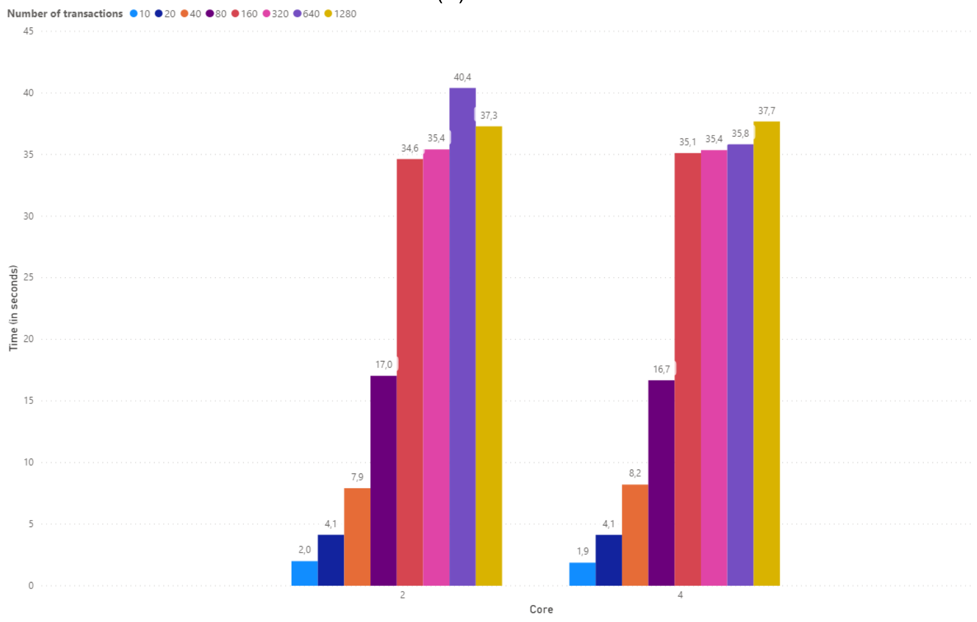
Figure 6.3: Scenario I - (1) attachToTangle computation times

that nodes take to complete tip selection, making them use more computational power. This behavior can be observed in Figure 6.4. Here, 800 and 1600 transactions were sent with a milestone depth of either 3 or 9 to a node and a proxy. In the previous benchmarks the milestone depth was of 3. Analysing the data, an increase in milestone depth seems to affect more the node in higher sent transactions whereas the proxy is more affected with less sent transactions. Nevertheless, the proxy outperforms the node overall.

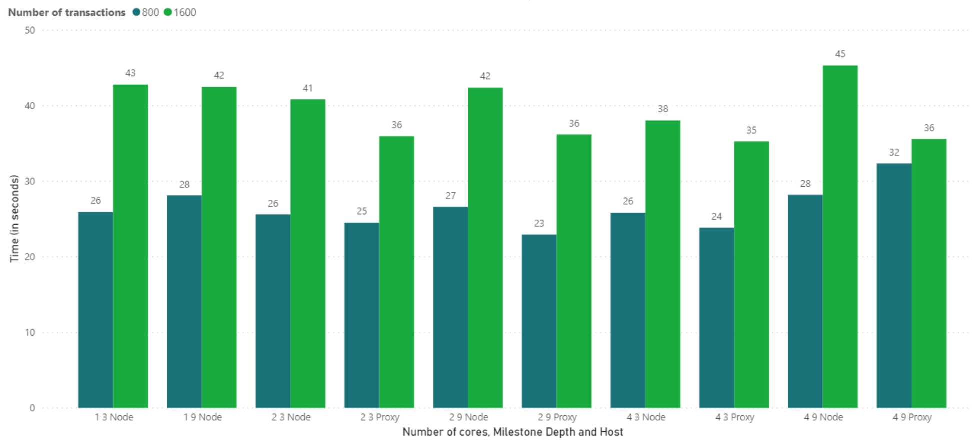
2. Transactions completed per second (Throughput): From the data gathered, in Figure 6.5, it is possible to observe that the throughput of the node and the proxy increase as the minimum weight magnitude of the network decreases for machines with four logical cores and a total of 1280 sent transactions. The average time to complete a transaction (Latency) follows the inverse path, it decreases when increasing the minimum weight magnitude. It is also possible to observe that the proxy can achieve higher throughput and lower latency. The highest throughput achieved was of



(a) Node



(b) Proxy



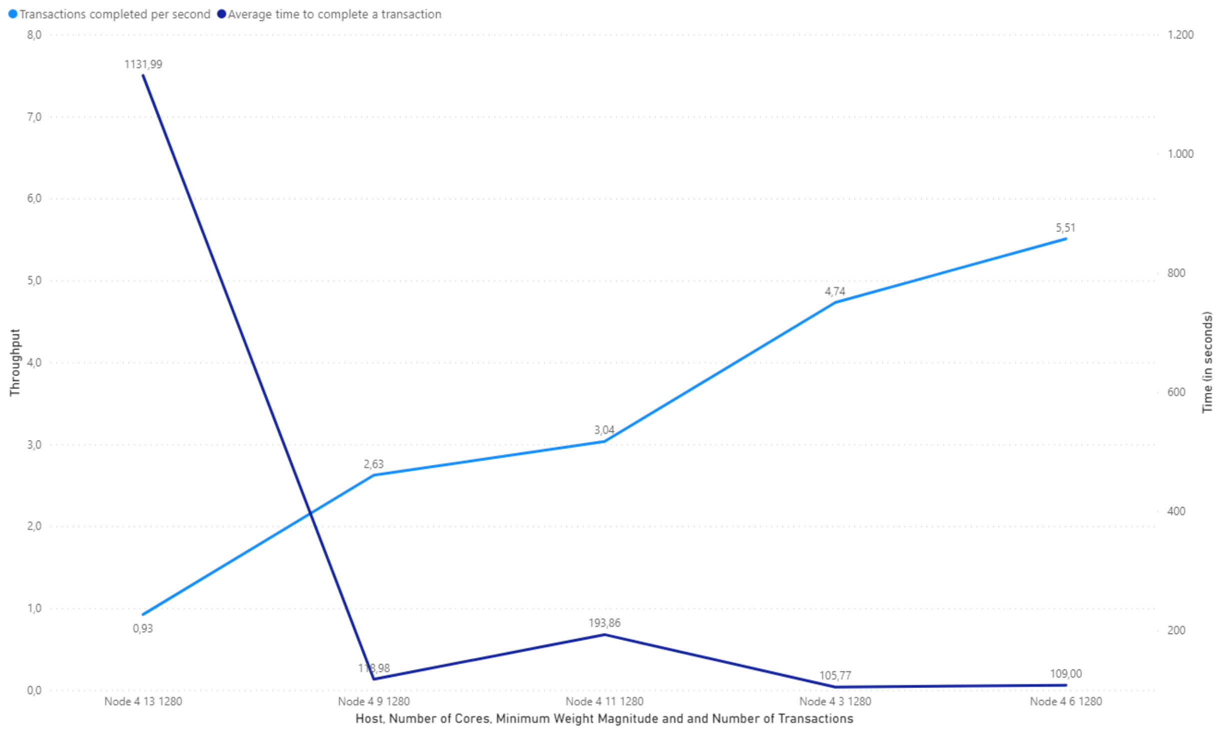
(c) Node and Proxy for different Milestone depths

Figure 6.4: Scenario I - (1) getTransactionsToApprove computation times

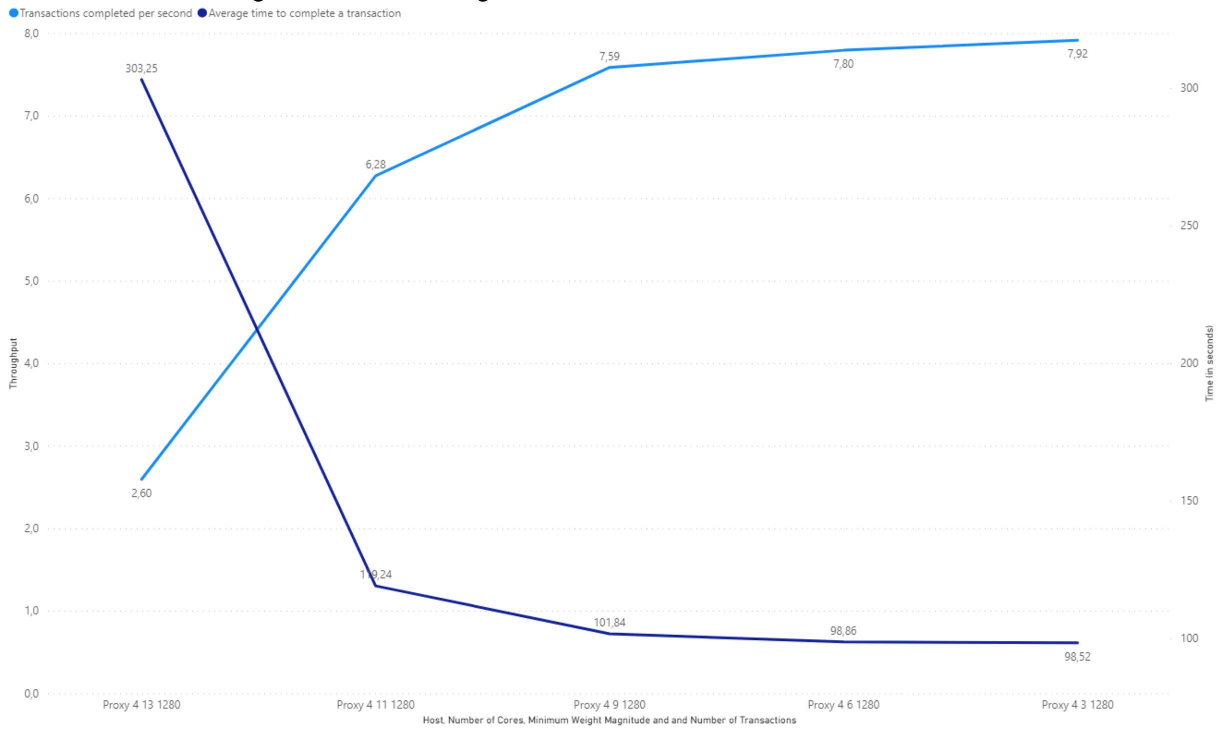
7.92 completed transactions per second, on a proxy of 4 logical cores, 3 as the minimum weight magnitude and 1280 total sent transactions. The lowest throughput achieved was of 0.16 completed transactions per second, on a node of 2 logical cores, 13 as the minimum weight magnitude and 320 total sent transactions. From this results, we can conclude that the minimum weight magnitude greatly influences the throughput and latency and that the proxy performs generally better than a node when receiving a high number of transactions. What about for a different number of logical cores?

3. Time to generate/complete transaction (Latency): From the data gathered, in Figure 6.6, we observe the variation in latency and time to send all the transactions for a different number of logical cores in both the node and the proxy. It is clear that the higher the number of cores, the lower the latency, for different numbers of sent transactions. The node with one logical processor has relatively the same performance as one node with two logical processors since parallelism can only be achieved starting from four logical processors. From Figure 6.6 (b), it is possible to observe how greatly the proxy performs when compared to a node with the same minimum weight magnitude and number of cores. But, what about for a different number of sent transactions? In Figure 6.7, it is possible to observe that the latency increases almost linearly with the number of sent transactions for each minimum weight magnitude and number of logical cores in general. However, a curious behavior can be found when analyzing this data in detail. From 10 to 80 sent transactions, the latency has a linear relationship, for both the node and the proxy and all minimum weight magnitudes, with the number of sent transactions, that is, if we multiply the number of sent transactions by 2, the latency will multiply by approximately 2. From 320 to 1280 sent transactions, the latency factor of growth will decrease or stay the same depending on the computational capacity of the machine or the minimum weight magnitude, that is, the trend is to have a factor growth rate of approximately 2 but this factor can decrease if the host is a proxy (which as shown to perform better than nodes) or the minimum weight magnitude is small enough or the number of logical cores is high enough. After this decrease, however, the growth factor can resume to the trend. To have a better perspective over this observations, the CPU and Memory usage will be analysed in 5.

4. Time to confirm transactions (Confirm Latency): From the data gathered, in Figure 6.8, we observe an histogram of the time to get a transaction approved by the coordinator with a rate of one milestone issued per minute (default of the IOTA network). It is clearly visible to what an extent a transaction can go to get approved by the coordinator. For the global values registered (for different logical cores, sent transactions and minimum weight magnitudes), the minimum value was of 0.5 seconds to approve, the maximum of 7482 seconds and it averaged around 654 seconds. It was also observed that the higher the number of transactions received by the coordinator, the higher the time to approve and standard deviation. During the tests, the coordinator software, compass, also faulted/crashed when the node was being overloaded with transactions. This could indicate that the coordinator is not sufficiently robust to perform under low computational resources or high network I/O. This caused some of the transactions to get even more delays to get approved.

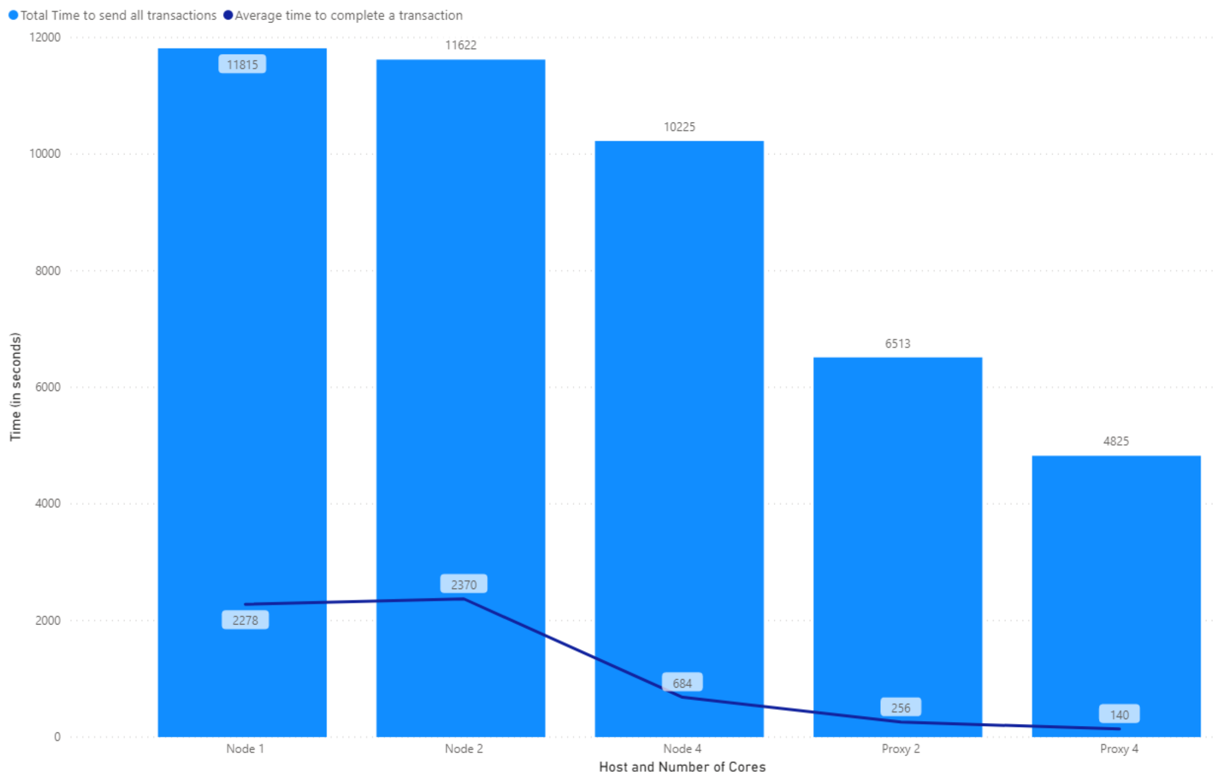


(a) Relationship between the Throughput and Latency of a Node for different Minimum Weight Magnitudes and the highest number of sent transactions and cores

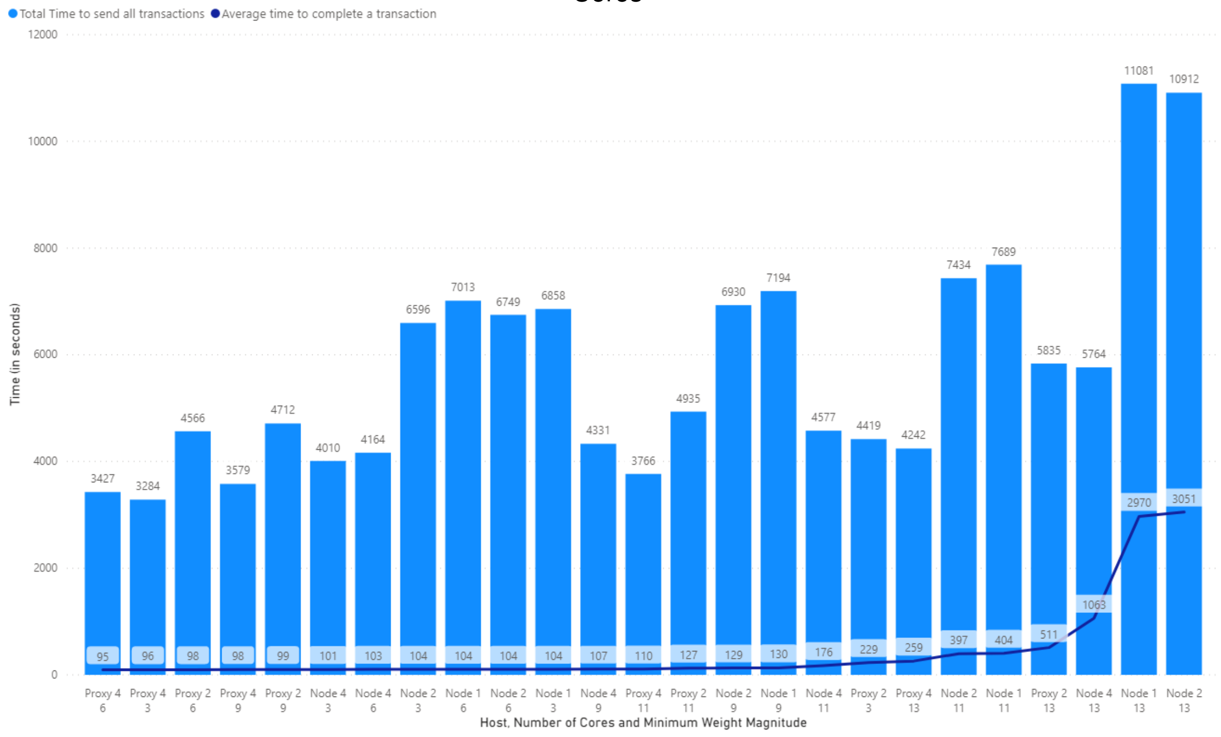


(b) Relationship between the Throughput and Latency of a Proxy for different Minimum Weight Magnitudes and the highest number of sent transactions and cores

Figure 6.5: Scenario I - (2) Throughput

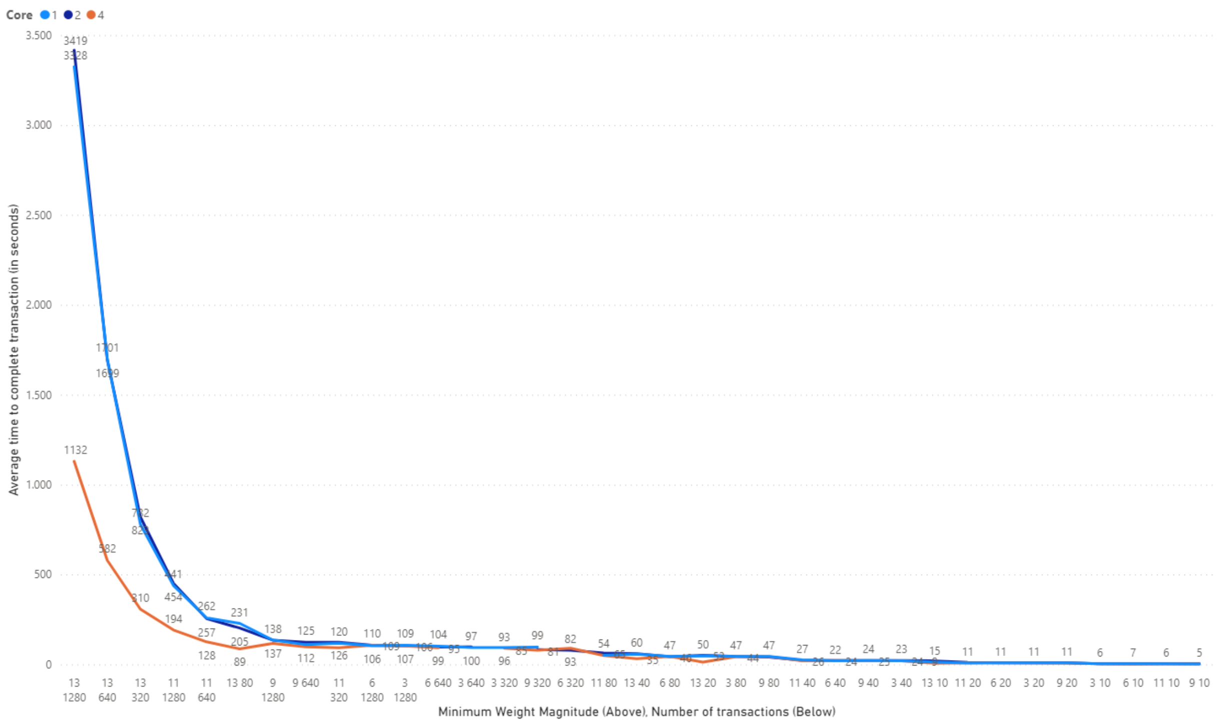


(a) Variation of latency and total time to send all transaction between Node and Proxy, and Number of Cores

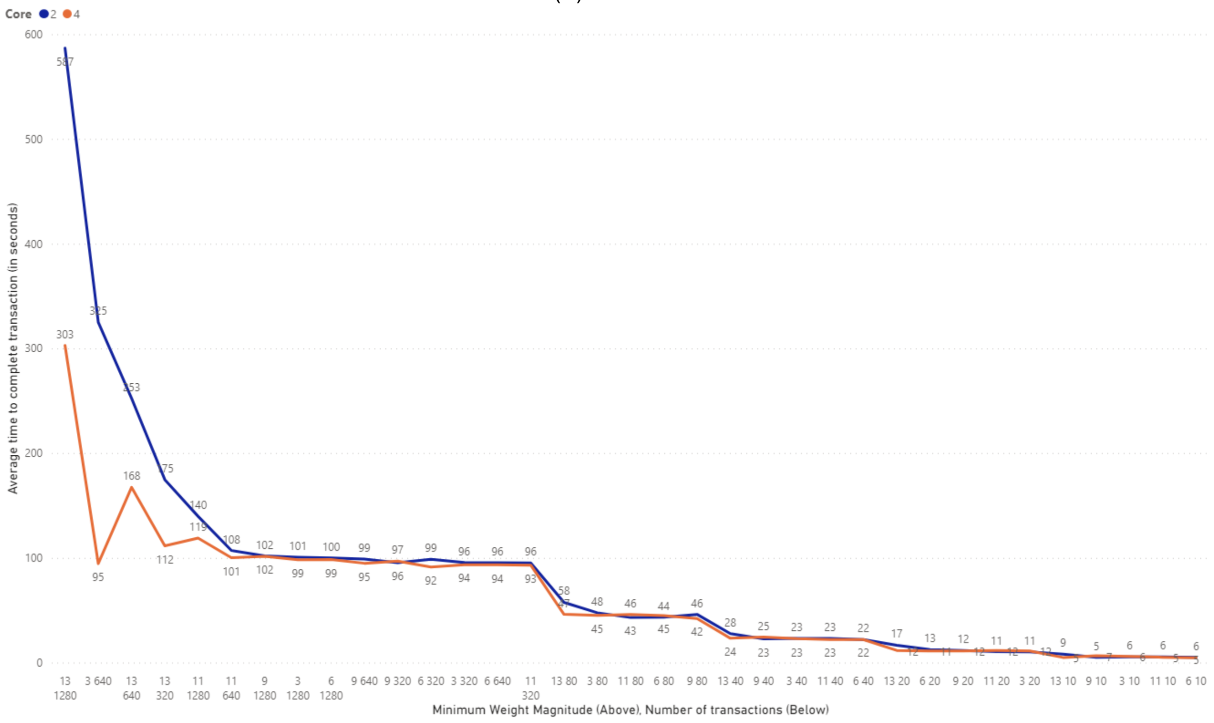


(b) Variation of latency and total time to send all transaction between Node and Proxy, Number of Cores and Minimum Weight Magnitude

Figure 6.6: Scenario I - (3) Latency

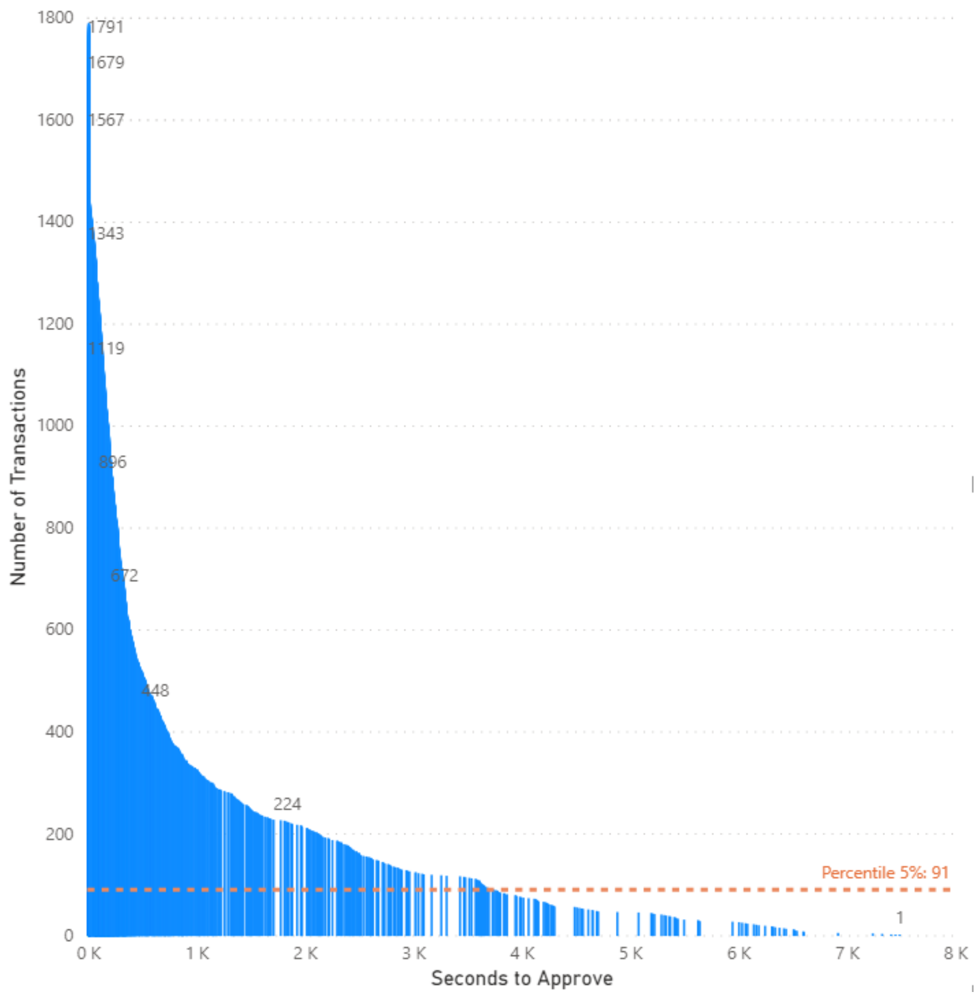


(a) Node

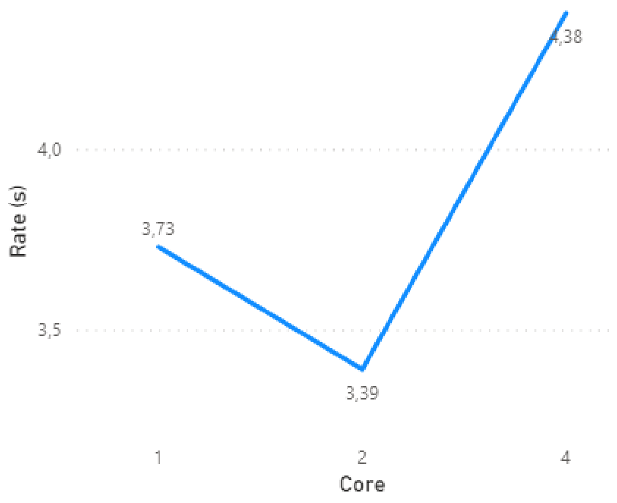


(b) Proxy

Figure 6.7: Scenario I - (3) Latency



(a) Histogram of time to approve a transaction



(b) Confirmed transactions per second on different logical cores

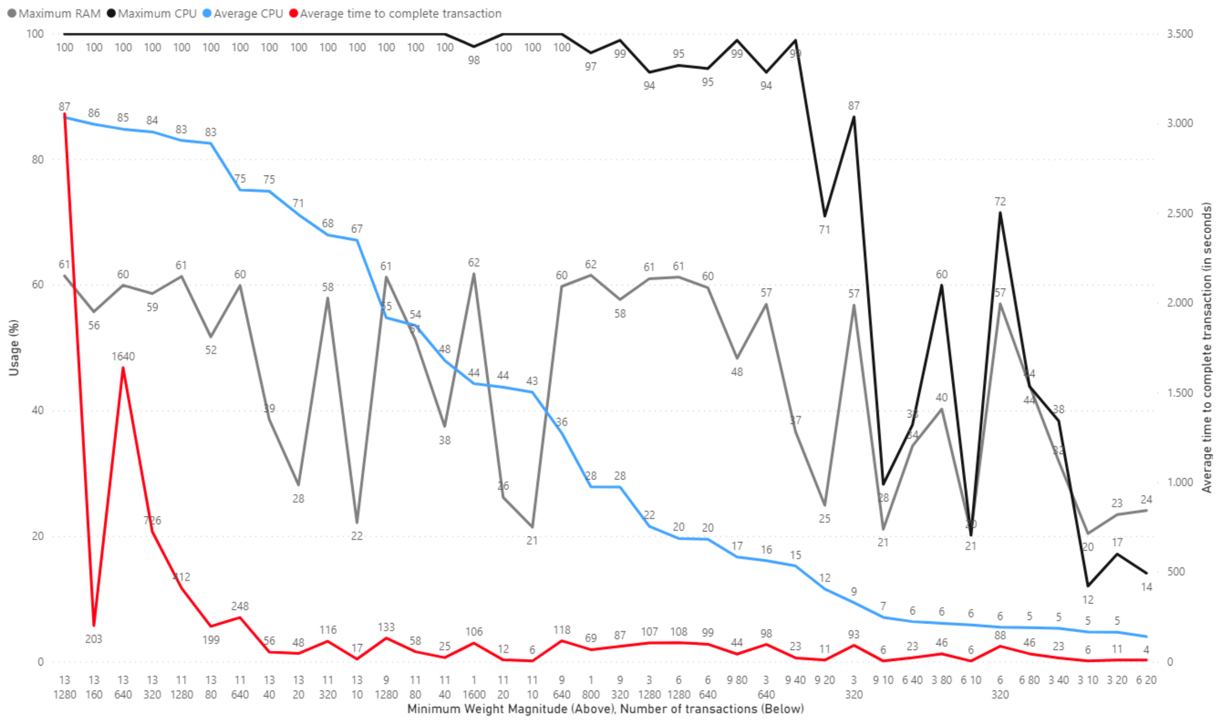
Figure 6.8: Scenario I - (4) Confirm Latency (in seconds)

5. CPU and Memory usage: From the data gathered, in Figure 6.9, the variation in CPU and Memory utilization of the node and the proxy for all different minimum weight magnitudes and transactions sent is observable. It presents an average result of all logical cores, to provide a general insight regardless of machine specifications. So, the first thing to notice is that, since the proxy only proxies the requests to a node and computes the proof-of-work, its memory utilization is very low, allowing us to know that the proof-of-work computation involves no memory utilization at all. The second thing to notice, is that the CPU utilization of the proxy almost never reaches 100% while receiving requests, meaning that it probably never got bottlenecked. The same behavior is not verified on the node, since it reaches 100% CPU utilization on considerably low numbers of sent transactions and minimum weight magnitudes. There is, however, nothing uncanny about this behavior, since it is known that the node is executing the IOTA protocol by running the IRI and Compass software giving it a disadvantage compared to a fully dedicated machine for computation. If we observe some of the latency results, for example for a minimum weight magnitude of 3 and 6 with 80 sent transactions, on both hosts prior to reaching a maximum registered value of 100% CPU utilization, we can see that the latency values and increase rate are similar. Observing them after hitting the max CPU utilization at a given time, the latency starts to increase by a lot. For example, for a minimum weight magnitude of 11 with 320, 640 and 1280 sent transactions, the node latency starts to increase at a different rate when compared to the proxy. In terms of memory usage on the node, it increases as more transactions are received. It, however, seems to reach an upper bound usage threshold at around 60% which was not possible to justify in the scope of this project.

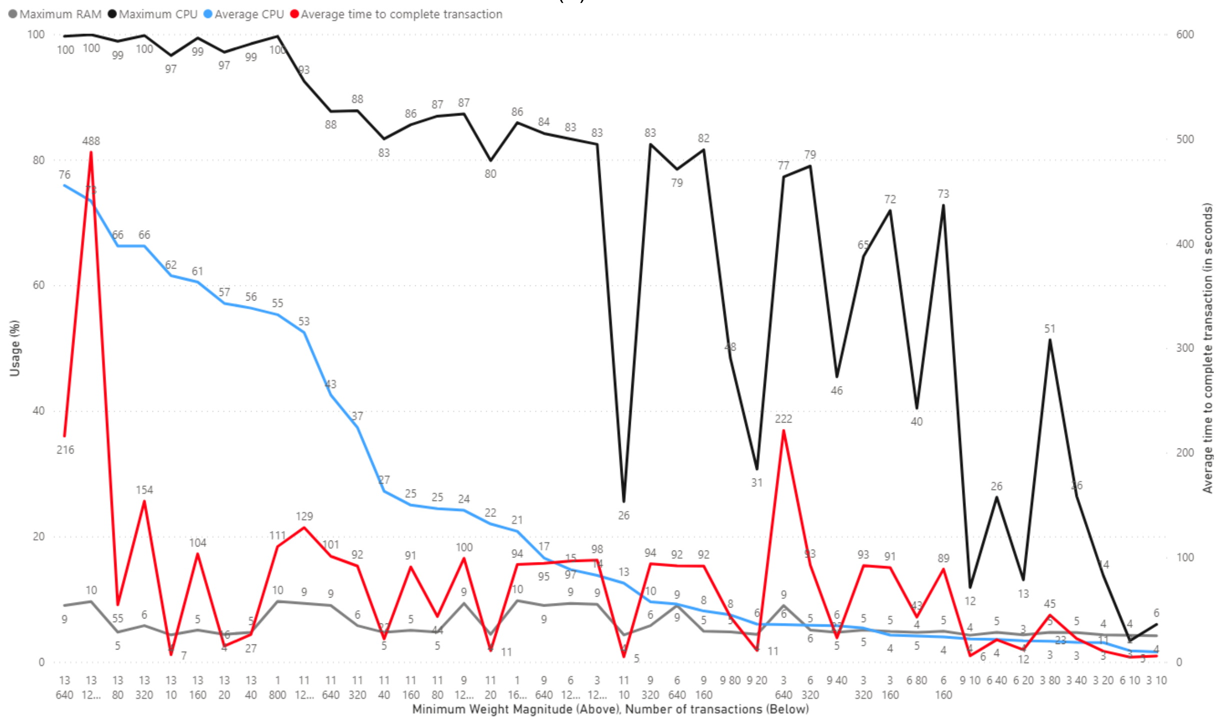
Scenario II

The Scenario II consist is one node and one proxies, both with 4 logical cores each, only a proof-of-work effort (Minimum weight magnitude) of 9 is studied and the network has an even higher workload (Higher number of sent transactions) when compared to Scenario I. This time, the tool sent a total of 1200, 1600, 1840, 2000, 2400, 3200 or 4000 transactions in total and the Throughput was limited by the number of transactions requested per second by the tool. The objective of this scenario was to extend the previous benchmarks presented in the Subsection 6.1.2 and understand how the node and the proxy would withstand an higher workload at a constant Throughput. Analysing the previously defined metrics, the following observations were produced:

1. Transactions completed per second (Throughput): From the data gathered, in Figure 6.10, it is possible to observe that the throughput remains relatively constant at around 5.5 to 6 transactions completed per second on the node and proxy, as expected. However, in the node, at around 2000 sent transactions a sudden drop in the throughput is visible. This was due to the fact that, at this point, the node started to not accept or drop transactions. This benchmark was repeated but it resulted in the same behavior. Now, was it caused by computational resources overload or network IO limitation? The discussion of this subject will continue in the next topic.



(a) Node

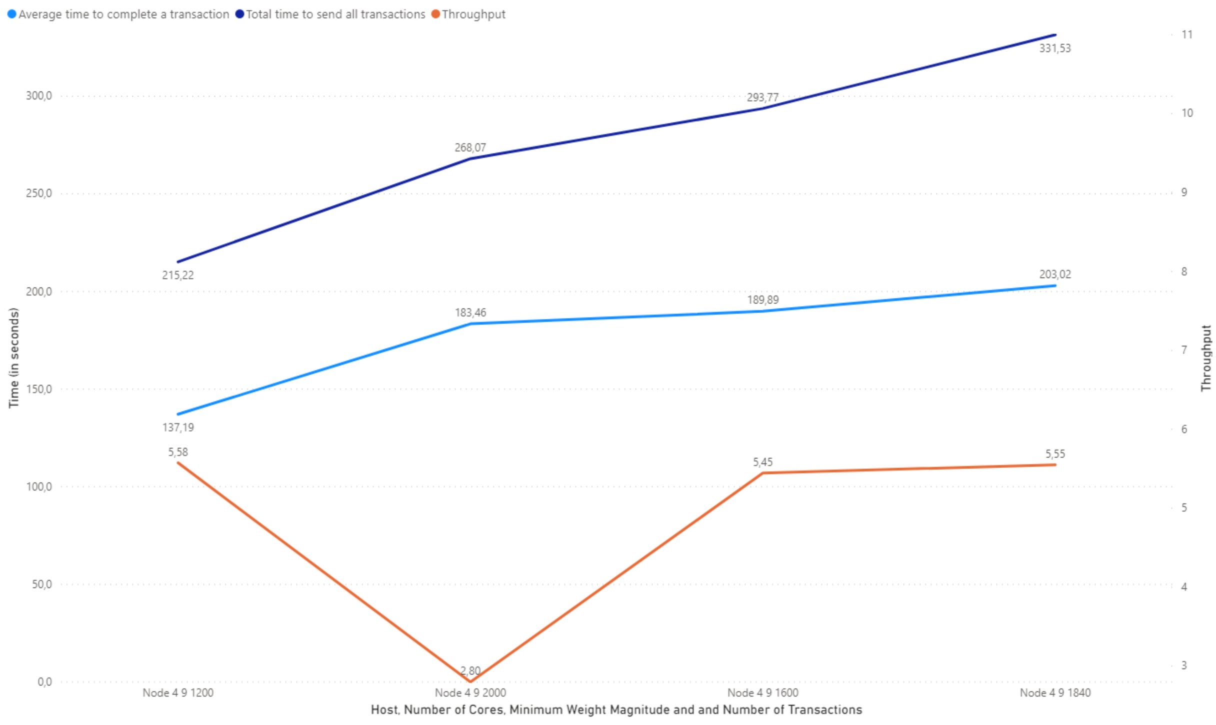


(b) Proxy

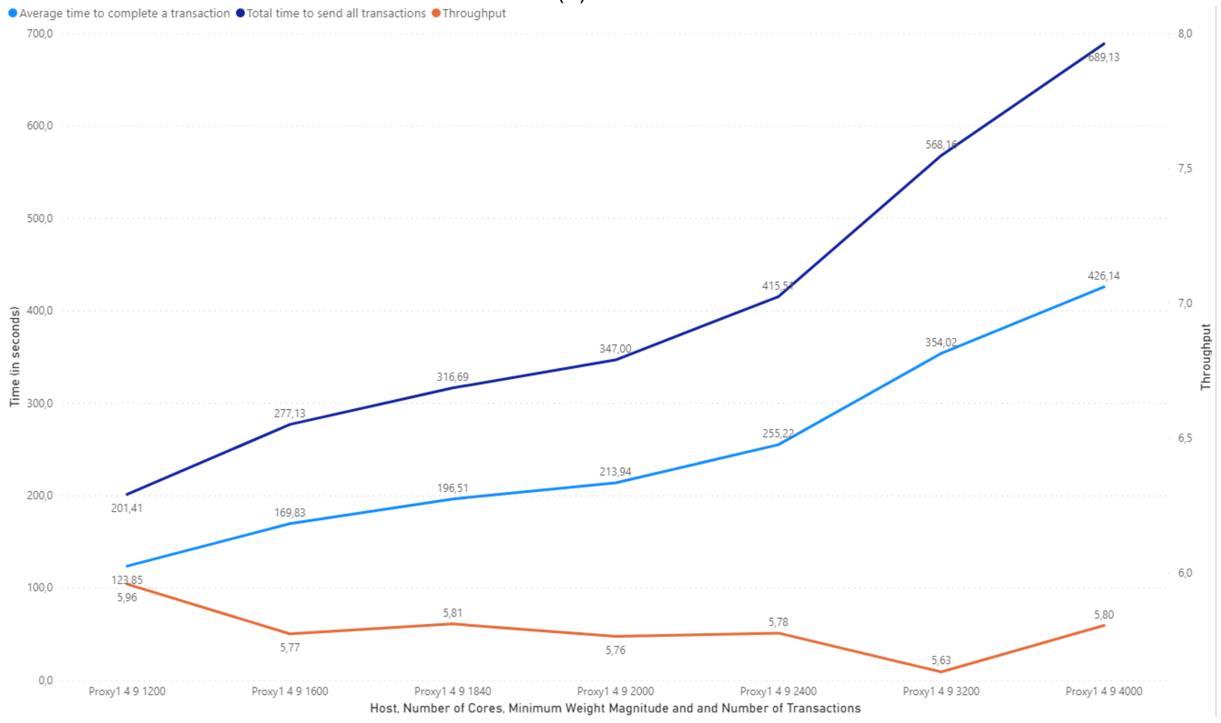
Figure 6.9: Scenario I - (5) CPU and Memory Usage

2. Node and Proxy comparison of CPU and Memory usage, Throughput and Time to generate/complete transaction (Latency): From the data gathered, in Figure 6.10, it is possible to observe the compiled data from the previously discussed metrics for both the node and proxy. In terms of computational resources, the average CPU utilization on both machines keeps relatively the same with some small variations, which makes sense since the throughput also is quite constant. The memory utilization, however, keeps increasing as more transactions are sent to the node. This probably means that at the time the node is receiving transactions from a benchmark, it still has not yet been able to process all transactions from the previous benchmark or was just not able to free its memory as the node might still be inserting transactions and other data in the database or trying to broadcast them to its neighbors. Resuming the subject discussed on the item 1, when the node faults at a total of 2000 sent transactions, the maximum and average registered CPU usage as well as the maximum and average Memory utilization are within reasonable values, nothing that could justify an overload of the system. Looking at the proxy, which is an identical machine, both in terms of resources and configurations, where the only difference is the running software that receives and processes the requests, it remains stable and within the expected results. This behavior is curious and uncanny because the proxy has to be always connected to a node in order to proxy all end user's requests except the proof-of-work computation. That said, the proxy was found to be more reliable and faster than a node when receiving requests from the end user's.

3. Time to confirm transactions (Confirm Latency): From the data gathered, in Figure 6.12, an histogram of the time to get a transaction approved by the coordinator with a rate of one milestone issued per minute (default of the IOTA network) is observable. For the global values registered (for different numbers of sent transactions), the minimum value was of 1.13 seconds to approve, the maximum of 6569 seconds and it averaged around 3491 seconds. It was also observed that the higher the number of transactions received by the coordinator, the higher the time to approve and standard deviation. Compared to the data obtained in the Scenario I, the histogram in the Scenario II is much more flat with transactions taking longer to get approved. During the tests, the coordinator software, compass, also faulted/crashed when the node was being overloaded with transactions. This behavior was also observed in the previous scenario. In conclusion, in the previous scenario it was seen that the coordinator took longer to approve transactions for an higher number of sent transactions, which by itself increased the throughput. Because the scenario consisted in testing a different number benchmarks for different minimum weight magnitudes, the node where the coordinator was running had more computational resources available between heavier benchmarks. In this scenario, the coordinator was running on almost the same resources thorough the benchmarks at an high throughput which might have lead to higher confirm latency. So, the confirm latency is, most likely, directly related to the throughput and computational resources usage over time.

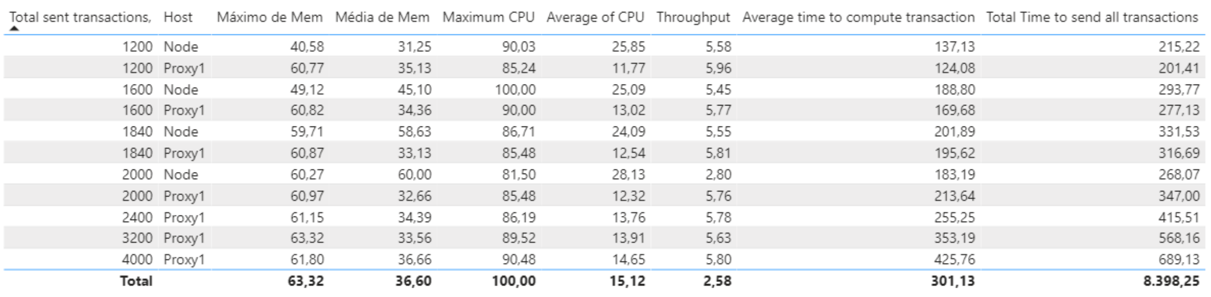


(a) Node



(b) Proxy

Figure 6.10: Scenario II - (1) Throughput



Total sent transactions,	Host	Máximo de Mem	Média de Mem	Maximum CPU	Average of CPU	Throughput	Average time to compute transaction	Total Time to send all transactions
1200	Node	40,58	31,25	90,03	25,85	5,58	137,13	215,22
1200	Proxy1	60,77	35,13	85,24	11,77	5,96	124,08	201,41
1600	Node	49,12	45,10	100,00	25,09	5,45	188,80	293,77
1600	Proxy1	60,82	34,36	90,00	13,02	5,77	169,68	277,13
1840	Node	59,71	58,63	86,71	24,09	5,55	201,89	331,53
1840	Proxy1	60,87	33,13	85,48	12,54	5,81	195,62	316,69
2000	Node	60,27	60,00	81,50	28,13	2,80	183,19	268,07
2000	Proxy1	60,97	32,66	85,48	12,32	5,76	213,64	347,00
2400	Proxy1	61,15	34,39	86,19	13,76	5,78	255,25	415,51
3200	Proxy1	63,32	33,56	89,52	13,91	5,63	353,19	568,16
4000	Proxy1	61,80	36,66	90,48	14,65	5,80	425,76	689,13
Total		**63,32**	**36,60**	**100,00**	**15,12**	**2,58**	**301,13**	**8.398,25**

Figure 6.11: Scenario II - (2): Node and Proxy comparison of CPU and Memory usage, Throughput and Latency

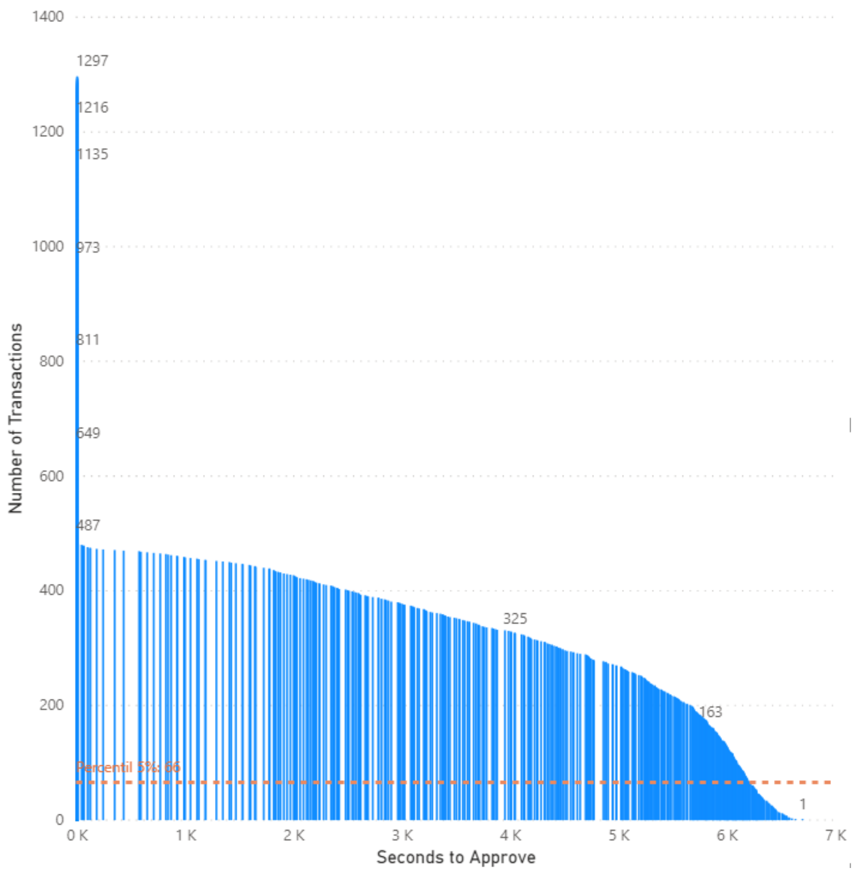


Figure 6.12: Scenario II - (3) Confirm Latency (in seconds)

6.2 Functional Tests

The validate the designed solution, the use cases defined in Section 4.5 were simulated. There were two types of operations to measure: (1) the Wallet Manager API operations and (2) the service provider endpoint operations. The first was tested using JUnit and filters over HTTP(s) requests that provided the execution times for each endpoint. The tests ran 50 times to get an average result and a more accurate representation. The error was gathered from the minimum and maximum execution times in the test. The second was tested automatically in the application with a simple loop function. It also gathered the results of 50 executions and averaged them, and the error was also measured in the same way. Therefore, since some of the created use cases use wallet manager API and service provider endpoint's operations, the results were then summed to create an approximation of the execution times in a real life scenario with only one client. The operations required for each use case can be found in Subsection 6.2.1 and the results in Subsection 6.2.2.

6.2.1 Use cases

The following list details the necessary operations required to execute in each use case:

US1 - Check Balance

1. Client selects Update Balance operation on the client application

US2 - Charge Wallet

1. Client selects Buy Tokens operation on the client application

US3 - Use Transport

1. Client selects Tickets operation on the client application
   (a) Client selects transport option 'Bus' on service provider 'MetroLx'
   (b) Client requests and receives ticket for 'Bus' on Service Provider 'MetroLx'
2. Client delivers the ticket to the service provider endpoint (metro gateway) and is ready to use the metro
3. Service provider endpoint (metro gateway) executes operation Receive Ticket

US4 - Use Transport (Wallet Consumption

1. Client selects Tickets operation on the client application
   (a) Client selects transport option 'Electrical Scooter' on service provider 'ScooterLx'
   (b) Client requests and receives check-in ticket for 'Electrical Scooter' on service provider 'ScooterLx'

Operation	Wallet Manager API endpoints	Average time to finish (ms)
Update Balance	GET /balance	13017.7
Buy Tokens	POST /buy	4486.55
Get price	GET /price	31
Get price based on distance	GET /uberprice	32.3
Generate Transaction	GET /transaction	1383.7
Generate Check-in transaction	GET /checkintransaction	1420.2
Generate Check-out transaction	GET /checkouttransaction	1585.1
Update Service Provider's Balance	GET /service/balance	15112.3
Update Service Provider's Price	PUT /service/updateprice	34.4
Update Service Provider's Address	PUT /service/updateaddress	278.2

Table 6.1: Wallet Manager endpoint's execution times

Operation	Average time to finish (ms)
Receive ticket	1721.2
Checkout ticket	3190.1

Table 6.2: Service endpoint operations execution times

2. Client delivers the check-in ticket to the service provider endpoint (scooter endpoint) and is ready to use the scooter

3. Service provider endpoint (scooter endpoint) executes operation Receive Ticket

4. Client uses the transport for 10 minutes

5. Client requests the check-out to the service endpoint (scooter endpoint)

6. Service provider endpoint (scooter endpoint) executes operation Checkout ticket

6.2.2 Experiment Results

The results obtained for the wallet manager API operations can be found in Table 6.1 and for the service provider endpoint in Table 6.2. The overall execution times for the use cases are shown in Figure 6.13. This results fit to the gathered results for the IOTA network in the Section 6.1 on a low number of total sent transactions. As a possible improvement, would be to replace the the Update Balance operation, on the wallet manager API, that requests the balance, transaction history and address information at the same time to multiple fined grained operations that not only allowed more control but also less execution times, as this operations was by far the longest to run.

6.3 Discussion

This sections justifies some of the design decisions made for this work by analysing three scenarios. We will consider, in the course of this section, as possible double-spending the scenario where a client

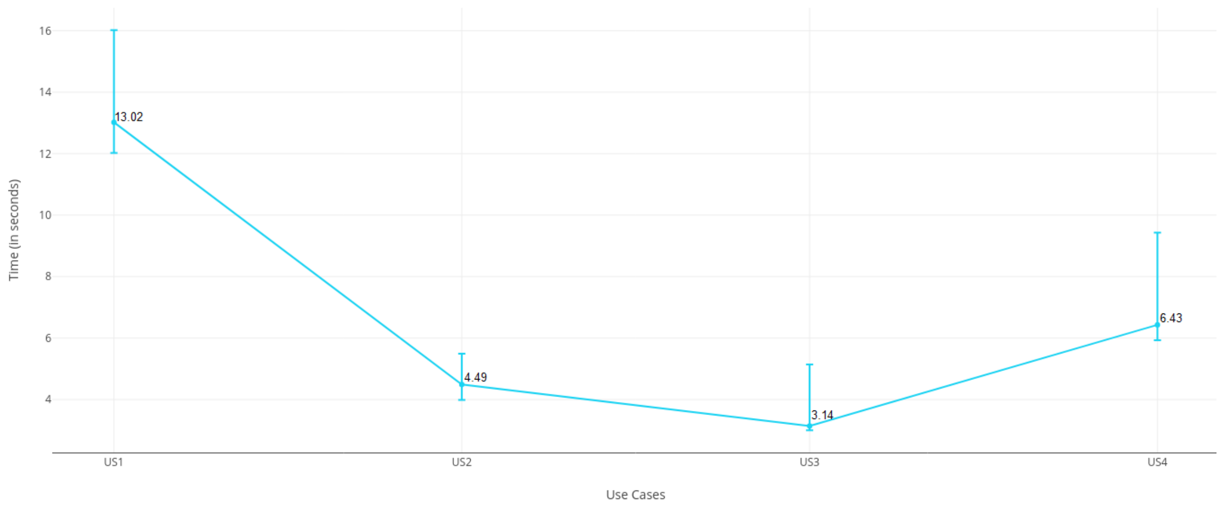


Figure 6.13: Average time to execute each Use Case

issues a fake ticket and is able to use a transport, even though the ticket contained in the transaction later becomes detected as a double-spend in the Tangle

1. Proof-of-work is computed at the transport gateways: In this scenario, the minimum weight magnitude can be a lower value than the standard IOTA network because the transactions can be generated by the gates instead of a client, which are less trusted – this results in a more centralized architecture which requires more computational resources. This architecture is not as centralized as the dedicated nodes alternative but it is not so feasible since the transportation gates are usually low-performance devices, resulting in a very high delay to compute proof-of-work (results can be found in Chapter 2 for a minimum weight magnitude of 14), thus making this scenario unfeasible.

2. Proof-of-work is computed at the client's smartphone: In this scenario, the minimum weight magnitude needs to be similar to the standard IOTA network because the transactions are generated by the client and it is used to prevent spamming of the network – this results in a more decentralized architecture making it possible to save network resources. However, because smartphones do not have the same computing power as a dedicated machine and the minimum weight magnitude needs to be high, this results in a high delay to compute proof-of-work (results can be found in Chapter 2 for a minimum weight magnitude of 14). The processing power of the device would also change the experience a lot from client to client and affect its battery. What also makes this scenario unfeasible is the possibility of double-spending. In this scenario the client would need to have in his possession all the necessary information to compute and generate a transaction and due to the time it could take to approve a transaction, the IOTA network would only detect a double-spend much time after the user had delivered the transaction.

3. Dedicated nodes or proxies to compute proof-of-work: In this scenario, the minimum weight magnitude can be a lower value than the standard IOTA network because the transactions can be generated by a dedicated trusted server instead of a client – this results in a more centralized and secure architecture which requires more computational resources but also makes it possible

to achieve better proof-of-work performance since the hosts can be high performance machines. It would also allow to implement a solution for the double-spending problem, by making sure that tickets were authentic and not allowing repeated tickets to enter the system. It was, as such, the most fit scenario to implement in a public transport ticketing system.

The previously obtained results, in Section 6.1, were conducted to evaluate the chosen scenario and to provide better insights when designing the solution. The main takeaways were:

1. Proof-of-work computation times highly depend on the chosen minimum weight magnitude
2. Tip selection computation times highly depend on number of transactions received by the host in a period of time, stabilizing after a certain threshold
3. Not enough computational resources on the proof-of-work computing host will cause a great delay in this operation
4. Node's seem to constantly grow its memory usage with time as more are sent transactions at a fixed throughput
5. A proxy is more reliable and better performing on high demanding environments (e.g., higher minimum weight magnitudes and/or higher number of requests)
6. Coordinator nodes constant high computational resources (CPU and/or Network IO) usage will cause a great delay in transaction confirmation speeds or even failure/crash

Gathering all this data and studying it for the chosen scenario, we could theoretically apply it to a real-life scenario. Using a proxy connected to a node, both with 4 logical processors (2 CPU's) and 4GB of RAM, to receive the requests from transport gateway/terminal at the same fixed rate as the previously executed benchmarks on a host with the equivalent specifications, a Throughput of 6 transactions completed per second would be accomplished. This means that a single proxy connected to a node would be able to send around 500.000 transactions to the Tangle per day. If we were to double its computational power to 8 logical cores (4 CPU's), we could also double its Throughput to 12 transactions completed per second and it would be possible to send around 1 million transactions to the Tangle. All this, while most likely not having more than 90% CPU usage at a given point in time. These results make the primary solution previously presented in Chapter 4 viable in terms of performance/cost relationship.

Chapter 7

Conclusions

In this chapter, it is described the main findings from this dissertation, as well as the existing limitations, and introduces some future work ideas.

7.1 Limitations

Despite all the advantages of using a DLT, such as IOTA, to implement a ticketing system in the mobility sector, there are still some limitations in this technology that were not addressed by its creators in the course of this work. The most important limitations of IOTA in this context were: (1) The high demanding computational effort required when computing the proof-of-work, for an effort level considered as the standard for a public network. This makes it difficult to integrate IOTA into low-performance or battery-dependent devices, (2) The coordinator as a single point of failure. This fact makes the confirmation of transactions reliant on a single node. If this node fails, the whole network is vulnerable to double-spending attacks and (3) The lack of performance and scalability benchmarks of its components and operations. This makes it difficult to design a network able to handle thousands of requests. As such, it was necessary to study or overcome these limitations.

7.2 Achievements

Given the defined requirement for processing micropayments in little time without much computational effort, IOTA promised to be the most interesting DLT to study. It was, at the start of the project, a relatively unknown and unstudied DLT. As such, in the Related Work chapter, IOTA was studied to deepen the technical knowledge and its applications in the IoT environment. Also, different DLTs were compared, aggregating the most positive and negative aspects of each solution. Some limitations found in the IOTA had to be addressed and to do so, some benchmarks were created and a solution was designed to mitigate some of the limitations in light of the requirements present in a ticketing system for the mobility sector. It was achieved by: (1) Reducing the proof-of-work effort, (2) Designing an overlayer which not only transformed IOTA into a permissioned network, preventing a client from travelling a number of times before its detected in the IOTA network as a double-spend, but also increased the resiliency of the coordinator and the scalability of the network and (3) Developing a series of benchmarks on IOTA's main components, the node and the proxy, in order to design a well-scaled network to support the developed solution and future works. With these problems addressed and understood, it was then possible to develop a solution as proof-of-concept of a ticketing system for multiple transport modalities and different usability schemes using IOTA. However, this design of the solution brought some other limitations which were not addressed in this work. In conclusion, despite having addressed some of the issues and having designed a workable proof-of-concept, we found IOTA 1.0 to be an going evolution not yet mature enough to be implemented in a large scale real life scenario.

7.3 Future Work

Benchmarks

To further improve the benchmarks executed on the IOTA components, we could distribute the client that is performing the requests so that more requests could be sent per second by not being limited to the computational resources of one machine. On the other hand, we could also increase the number of nodes in the IOTA network to study its scalability. Benchmarks could be also executed on low-performance devices, fit for IoT, on lower proof-of-work efforts.

Solution

To further improve the solution, the wallet manager component needed to be benchmarked and audited in order to transform it into a secure and highly available component, since without it would be impossible to use the network. An other component could also be added to the solution to provide better auditing capabilities. As a proof-of-concept, we could distribute the designed components into different networks and implemented the service provider endpoint in a low-performance device (e.g., Raspberry Pi), which would allow a more seamless and real-life experience.

IOTA

To further improve IOTA 1.0, the IOTA foundation has proposed in January 2020 a series of improvements to what they called Coordicide [76], which would finally bring IOTA to a place closer to its objectives. Some of this new changes are now available to study and test since they are implemented in new components, but were not, however, until very recently, not making it in time for this project. We believe that with this change, IOTA could be the many steps closer to the DLT that was initially promised for processing micropayments in little time without much computational effort in IoT devices. When all of this new changes get implemented, IOTA will release its version 2.0. Some of the new changes include:

- Node accountability: the concept of global node IDs and a novel Sybil protection mechanism that does not require node owners to risk or disclose their funds. Identifying the issuing node of a message is fundamental to enforce a specific network topology (through autopeering) or to penalize bad behaviours (through rate control).

- Autopeering and node discovery: An automated process to discover and reliably connect to neighbors is needed in every distributed system.
- Rate control: To ensure the network does not exceed its capacity, a mechanism to control the rate of transactions that are propagated through the network. This method selectively filters some transactions out according to the statistics of the issuing node.
- Consensus and Voting: The previous building blocks lead to an extended consensus framework. The new protocol does not use the tip selection algorithm as a tool for consensus, although it is still has its importance. Instead, it proactively resolves conflicts through voting. Two voting mechanisms were proposed where nodes query other nodes to find out their current opinion on the network status.
- Tip Selection: With the decoupling from the consensus mechanism, the tip selection algorithm has more freedom to achieve a better performance in its other tasks, as keeping the good structure of the network, demotivating lazy behavior and quickly approve new honest transactions. More details were proposed for a new tip selection. algorithm.

Bibliography

[1] P. Coelho. *Internet Das Coisas - Introduçao Pratica*. FCA. ISBN 9789727228492. URL `https://books.google.pt/books?id=g2JptAEACAAJ`.

[2] A. Dorri, S. S. Kanhere, and R. Jurdak. Towards an optimized blockchain for iot. In *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, IoTDI '17, page 173–178, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349666. doi: 10.1145/3054977.3055003. URL `https://doi.org/10.1145/3054977.3055003`.

[3] C. Fan. Performance analysis and design of an iot-friendly dag-based distributed ledger system. 2019.

[4] J.-S. Coron and J. Nielsen. *Advances in Cryptology – EUROCRYPT 2017: 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 – May 4, 2017, Proceedings, Part II*, volume 10211. 01 2017. ISBN 978-3-319-56613-9. doi: 10.1007/978-3-319-56614-6.

[5] S. El-Hage and G. Holst. Micropayments between iot devices: A qualitative study analyzing the usability of dlt: s in an iot environment, 2018.

[6] C. Prableen Bajpai. Distributed ledger technology, 2018. URL `https://www.investopedia.com/terms/d/distributed-ledger-technology-dlt.asp`.

[7] S. Popov. The tangle. 2018.

[8] B. A. E. Report. Cryptocurrencies: looking beyond the hype. 2018.

[9] A. Berentsen and F. Schär. A short introduction to the world of cryptocurrencies. *Federal Reserve Bank of St Louis*, 100(1), 2018. Review.

[10] E. Team. 10 facets of blockchain. *PTI*, (1):13–15, 2018.

[11] I. T. F. Philippe Crist. Blockchain and beyond: Encoding 21st century transport. 2018.

[12] P. Jittrapirom, V. Caiati, A.-M. Feneri, S. Ebrahimigharehbaghi, M. A. González, and J. Narayan. Mobility as a service: A critical review of definitions, assessments of schemes, and key challenges. 2017.

[13] Econotimes. Blockchain project antshares explains reasons for choosing dbft over pow and pos, 2017. URL `https://www.econotimes.com/Blockchain-project-Antshares-explains-reasons-for-choosing-dBFT-over-PoW-and-PoS-659275`.

[14] X. N. W. T. H. Ren, C.; Lyu and R. P. Liu. Distributed online learning of fog computing under non-uniform device cardinality. *IEEE Internet of Things Journal*, (1-1), 2018.

[15] X. N. W. L. R. G. Y. N. X. Z. K. Wang, X.; Zha. Survey on blockchain for internet of things. *Computer Communications*, 2019.

[16] G. Hileman and M. Rauchs. 2017 global blockchain benchmarking study. 2017.

[17] J. Ihrig, E. Meade, and G. Weinbach. Finance and economics discussion series divisions of research statistics and monetary affairs federal reserve board, washington, d.c. *Finance and Economics Discussion Series*, 2015, 06 2015. doi: 10.17016/FEDS.2015.047.

[18] C. O'Connor. What blockchain means for you, and the internet of things, 2017. URL `https://www.ibm.com/blogs/internet-of-things/watson-iot-blockchain/`.

[19] C. C. J. S. E. D. M. Reyna, A.; Martín. On blockchain and its integration with iot. challenges and opportunities. *Future Generation Computer Systems*, 88:173–190, 2018.

[20] O. Vashchuk and R. Shuwar. Pros and cons of consensus algorithm proof of stake. difference in the network safety in proof of work and proof of stake. *Electronics and Information Technologies*, 9, 01 2018. doi: 10.30970/eli.9.106.

[21] M. Maroufi, R. Abdolee, and B. Mozaffari Tazehkand. On the convergence of blockchain and internet of things (iot) technologies. 03 2019. doi: 10.33423/jsis.v14i1.990.

[22] R. P. M. Lamport, L.; Shostak. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 1982.

[23] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, page 173–186, USA, 1999. USENIX Association. ISBN 1880446391.

[24] C. Cachin and M. Vukolić. Blockchain consensus protocols in the wild, 2017.

[25] S. King and S. Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. 2012.

[26] K. Thulasiraman and M. Swamy. Graphs: Theory and algorithms. pages i–xvi, 02 2011. doi: 10.1002/9781118033104.fmatter.

[27] K. Košťál, T. Krupa, M. Gembec, I. Vereš, M. Ries, and I. Kotuliak. On transition between pow and pos. In *2018 International Symposium ELMAR*, pages 207–210, 2018.

[28] Q. Lin, H. Yan, Z. Huang, W. Chen, J. Shen, and Y. Tang. An id-based linearly homomorphic signature scheme and its application in blockchain. *IEEE Access*, 6:20632–20640, 2018.

[29] X. L. J. S. J. Y. Y. Wang, L.; Shen. Cryptographic primitives in blockchains. *Journal of Network and Computer Applications*, 2018.

[30] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang. Untangling blockchain: A data processing view of blockchain systems. *IEEE Transactions on Knowledge and Data Engineering*, 30(7):1366–1385, 2018.

[31] Bitcoin energy consumption index. URL `https://digiconomist.net/bitcoin-energy-consumption/`.

[32] C. Fan, H. Khazaei, Y. Chen, and P. Musilek. Towards a scalable dag-based distributed ledger for smart communities. In *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, pages 177–182, 2019.

[33] R. Kastelein. Intel jumps into blockchain technology storm with 'sawtooth lake' distributed ledger, 2016. URL `https://www.the-blockchain.com/2016/04/09/intel-jumps-into-blockchain-technology-storm-with-sawtooth-lake-distributed-ledger/`.

[34] Y. Yuan and H. Zhiwei. Dag technology analysis and measurement. 2018.

[35] K. Yeow, A. Gani, R. W. Ahmad, J. J. P. C. Rodrigues, and K. Ko. Decentralized consensus for edge-centric internet of things: A review, taxonomy, and research issues. *IEEE Access*, 6:1513–1524, 2018.

[36] D. Strugar, R. Hussain, M. Mazzara, V. Rivera, J. Young Lee, and R. Mustafin. On m2m micropayments: A case study of electric autonomous vehicles. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1697–1700, 2018.

[37] R. Radhakrishnan and B. Krishnamachari. Streaming data payment protocol (sdpp) for the internet of things. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1679–1684, 2018.

[38] P. C. Bartolomeu, E. Vieira, and J. Ferreira. Iota feasibility and perspectives for enabling vehicular applications. In *2018 IEEE Globecom Workshops (GC Wkshps)*, pages 1–7, 2018.

[39] A. Elsts, E. Mitskas, and G. Oikonomou. Distributed ledger technology and the internet of things: A feasibility study. pages 7–12, 11 2018. doi: 10.1145/3282278.3282280.

[40] I. Community. An introduction to iota. URL `https://www.helloiota.com/articles/back-to-the-basics`.

[41] IOTA. Kerl, . URL `https://github.com/iotaledger/kerl`.

[42] M. P. G. V. A. G. Bertoni, J. Daemen and R. V. Keer. Keccak implementation overview. 2012. URL https://keccak.team/files/Keccak-implementation-3.2.pdf.

[43] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In A. Miri and S. Vaudenay, editors, *Selected Areas in Cryptography*, pages 320–337, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[44] IOTA. Iota: What is a transaction?, . URL https://docs.iota.org/docs/getting-started/0.1/transactions/transactions.

[45] H. Sukhwani, N. Wang, K. S. Trivedi, and A. Rindos. Performance modeling of hyperledger fabric (permissioned blockchain network). In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–8, 2018.

[46] G. Becker and R. universität Bochum. Merkle signature schemes, merkle trees and their cryptanalysis, 2008.

[47] P. S. Bartosz Kusmierz and A. Gal. Extracting tangle properties in continuous time via large-scale simulations. 2018.

[48] A. Back. Hashcash - a denial of service counter-measure. 09 2002.

[49] L. de Vries. Iota vulnerability: Large weight attack performed in a network, January 2019. URL http://essay.utwente.nl/77602/.

[50] A. Cullen, P. Ferraro, C. King, and R. Shorten. Distributed ledger technology for iot: Parasite chain attacks. 03 2019.

[51] G. De Roode, I. Ullah, and P. J. M. Havinga. How to break iota heart by replaying? In *2018 IEEE Globecom Workshops (GC Wkshps)*, pages 1–7, 2018.

[52] Xy attack vector - iotas version of the 34% attack. URL https://forum.helloiota.com/469/XY-Attack-Vector-IOTAs-version-of-the-34-attack.

[53] Docker. URL https://www.docker.com/.

[54] Iota iri. URL https://github.com/iotaledger/iri.

[55] Rocksdb. URL https://rocksdb.org/.

[56] Caddy. URL https://caddyserver.com/.

[57] Compass. URL https://github.com/iotaledger/compass.

[58] Vmware. URL https://www.vmware.com/.

[59] Spring boot, . URL https://spring.io/projects/spring-boot.

[60] Sqlite. URL https://www.sqlite.org/index.html.

[61] Spring framework, . URL `https://spring.io/projects/spring-framework`.

[62] Junit. URL `https://junit.org/`.

[63] Jota. URL `https://github.com/iotaledger/iota-java`.

[64] Ionic framework. URL `https://ionicframework.com/`.

[65] Angular. URL `https://angular.io/`.

[66] React. URL `https://reactjs.org/`.

[67] Vue.js. URL `https://vuejs.org/`.

[68] Raspberry pi. URL `https://www.raspberrypi.org/`.

[69] Pyota. URL `https://github.com/iotaledger/iota.py`.

[70] Python global interpreter lock. URL `https://realpython.com/python-gil/`.

[71] asyncio - python library. URL `https://docs.python.org/3/library/asyncio.html`.

[72] multiprocessing - python library. URL `https://docs.python.org/3/library/multiprocessing.html`.

[73] Zeromq. URL `https://zeromq.org/`.

[74] top - linux command. URL `https://linux.die.net/man/1/top`.

[75] Power bi. URL `https://powerbi.microsoft.com/`.

[76] Coordicide. URL `https://files.iota.org/papers/20200120_Coordicide_WP.pdf`.

Appendix A

Main types of blockchain systems

	Read	**Write**	**Commit**
Public permissionless	Open to anyone	Anyone	Anyone
Public permissioned	Open to anyone	Authorized participants	All or subset of authorized participants
Consortium	Restricted to an authorized set of participants	Authorized participants	All or subset of authorized participants
Private permissioned	Fully private or restricted to a limited set of authorized nodes	Network operator only	Network operator only

Table A.1: Main types of blockchain systems

Appendix B

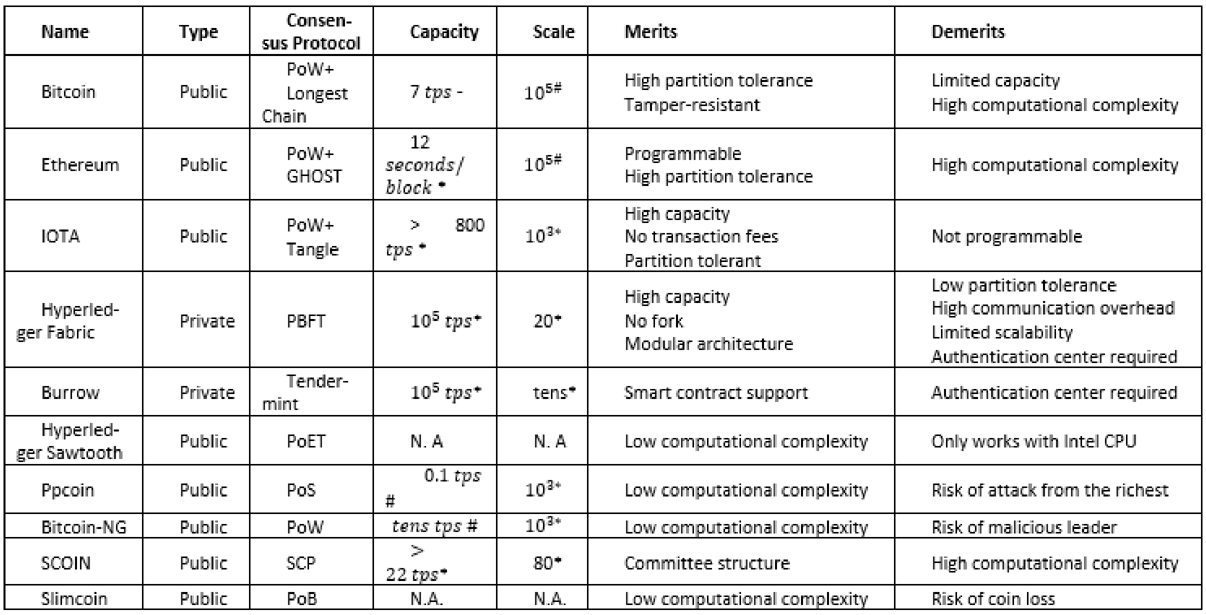
Comparison of Consensus Protocols

Consensus Protocol	**Network Settings**	**Description**
PBFT-based	Private	Hyperledger uses the original PBFT. Tendermint enhances it by assigning unequal weights to votes. Other variants include Scalable BFT, Parallel BFT, Optimistic BFT, etc.
Stellar	Federated	Stellar Network proposes its own consensus protocol where the nodes form intersecting groups (federates). Consensus is agreed in each group, then propagated to the rest of the net-work.
Ripple	Federated	Ripple payment system proposes a variant of PBFT where the nodes belong to intersecting groups, and in each group, there is a large major-ity of non-Byzantine nodes.
Proof-of-Work (PoW)	Public	Bitcoin uses pure proof-of-work, which leads to scalability issues. Bitcoin-NG, Byzcoin separate leader election from transaction validation in PoW, thus increase the overall performance.
Proof-of-Stake (PoS)	Public	Tendermint uses PoS, in which a node's ability to create net block is determined by its stake in the blockchain, e.g. the amount of currencies it owns. A set of high-stake owners uses another consensus mechanism, which is usually faster than PoW, to reach agreement on a new block.
Proof-of-Authority (PoA)	Private	Parity uses PoA, in which some predefined nodes are considered trusted authorities and they can propose the next blocks. It then uses round-robin scheduling to assign every authority node a time window during which it can propose blocks.
Proof-of-Elapsed Time (PoET)	Private	Sawtooth uses PoET, in which each node runs a trusted hardware, for example Intel SGX, that generates random timers. The first node whose timer has expired can propose the next block.
Proof-of-Burn (PoB)	Public	Slimcoin uses PoB, in which a node destroys some base currencies it owns in another blockchan in order to get a chance of proposing a new block. Slimcoin supports PoB based on Peercoin.
Proof-of-Elapsed Time (PoET)	Private	Sawtooth uses PoET, in which each node runs a trusted hardware, for example Intel SGX, that generates random timers. The first node whose timer has expired can propose the next block.
Others	Public	Other protocols based on PoW are of the form proof-of-X, for examples: Proof-of-Activity, Proof-of-Space, Proof-of-Luck, etc. Other non-PoW based consensus protocols include the Tangle, used in IOTA.

Table B.1: Comparison of Consensus Protocols

Appendix C

Performance Comparison of Blockchain in IoT application

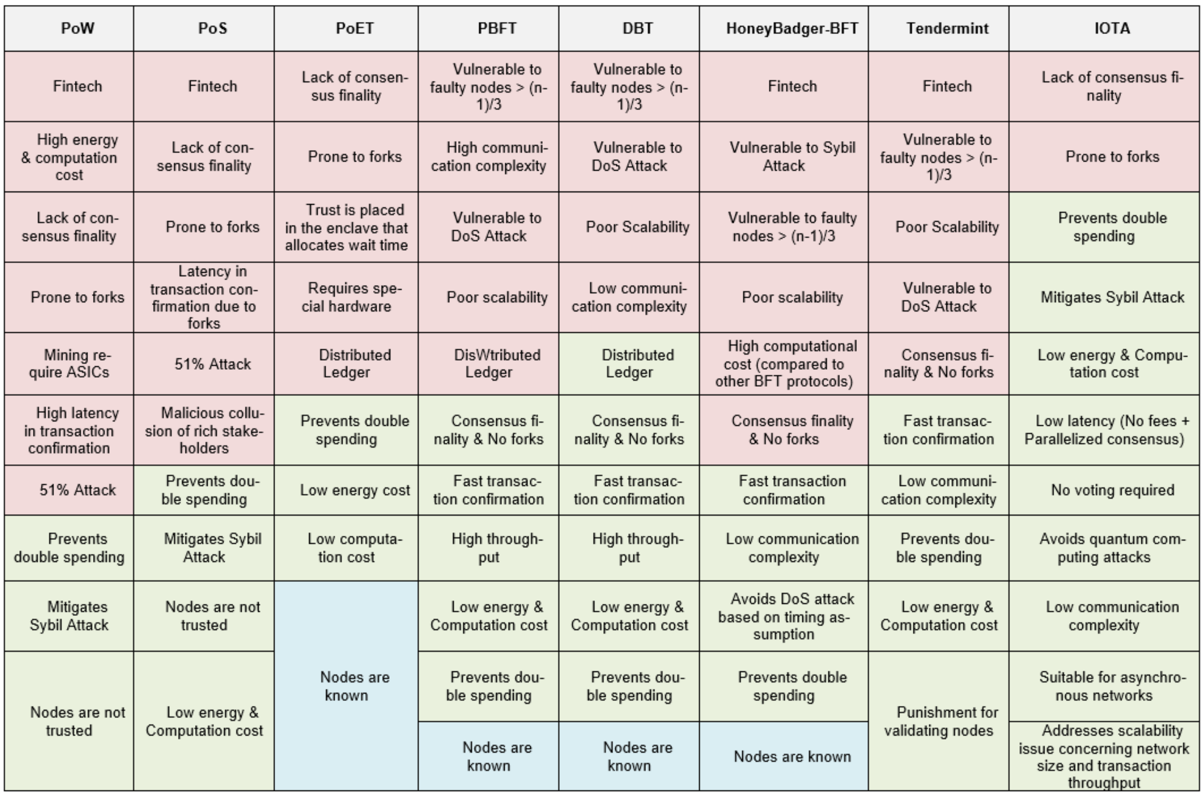


Name	Type	Consen-sus Protocol	Capacity	Scale	Merits	Demerits
Bitcoin	Public	PoW+ Longest Chain	7 *tps* -	$10^{5\#}$	High partition tolerance Tamper-resistant	Limited capacity High computational complexity
Ethereum	Public	PoW+ GHOST	12 *seconds/ block* *	$10^{5\#}$	Programmable High partition tolerance	High computational complexity
IOTA	Public	PoW+ Tangle	> 800 *tps* *	10^{3*}	High capacity No transaction fees Partition tolerant	Not programmable
Hyperled-ger Fabric	Private	PBFT	10^5 *tps**	20*	High capacity No fork Modular architecture	Low partition tolerance High communication overhead Limited scalability Authentication center required
Burrow	Private	Tender-mint	10^5 *tps**	tens*	Smart contract support	Authentication center required
Hyperled-ger Sawtooth	Public	PoET	N. A	N. A	Low computational complexity	Only works with Intel CPU
Ppcoin	Public	PoS	0.1 *tps* #	10^{3*}	Low computational complexity	Risk of attack from the richest
Bitcoin-NG	Public	PoW	*tens tps* #	10^{3*}	Low computational complexity	Risk of malicious leader
SCOIN	Public	SCP	> 22 *tps**	80*	Committee structure	High computational complexity
Slimcoin	Public	PoB	N.A.	N.A.	Low computational complexity	Risk of coin loss

Figure C.1: Performance Comparison of Blockchain in IoT application

Appendix D

Comparison of Consensus Protocols

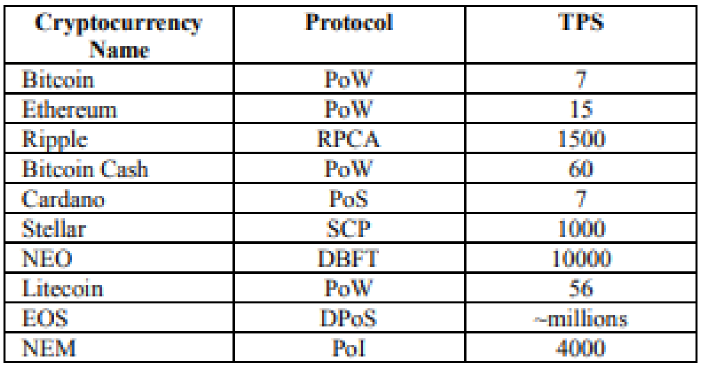


PoW	PoS	PoET	PBFT	DBT	HoneyBadger-BFT	Tendermint	IOTA
Fintech	Fintech	Lack of consensus finality	Vulnerable to faulty nodes > (n-1)/3	Vulnerable to faulty nodes > (n-1)/3	Fintech	Fintech	Lack of consensus finality
High energy & computation cost	Lack of consensus finality	Prone to forks	High communication complexity	Vulnerable to DoS Attack	Vulnerable to Sybil Attack	Vulnerable to faulty nodes > (n-1)/3	Prone to forks
Lack of consensus finality	Prone to forks	Trust is placed in the enclave that allocates wait time	Vulnerable to DoS Attack	Poor Scalability	Vulnerable to faulty nodes > (n-1)/3	Poor Scalability	Prevents double spending
Prone to forks	Latency in transaction confirmation due to forks	Requires special hardware	Poor scalability	Low communication complexity	Poor scalability	Vulnerable to DoS Attack	Mitigates Sybil Attack
Mining require ASICs	51% Attack	Distributed Ledger	DisWtributed Ledger	Distributed Ledger	High computational cost (compared to other BFT protocols)	Consensus finality & No forks	Low energy & Computation cost
High latency in transaction confirmation	Malicious collusion of rich stakeholders	Prevents double spending	Consensus finality & No forks	Consensus finality & No forks	Consensus finality & No forks	Fast transaction confirmation	Low latency (No fees + Parallelized consensus)
51% Attack	Prevents double spending	Low energy cost	Fast transaction confirmation	Fast transaction confirmation	Fast transaction confirmation	Low communication complexity	No voting required
Prevents double spending	Mitigates Sybil Attack	Low computation cost	High throughput	High throughput	Low communication complexity	Prevents double spending	Avoids quantum computing attacks
Mitigates Sybil Attack	Nodes are not trusted	Nodes are known	Low energy & Computation cost	Low energy & Computation cost	Avoids DoS attack based on timing assumption	Low energy & Computation cost	Low communication complexity
Nodes are not trusted	Low energy & Computation cost		Prevents double spending	Prevents double spending	Prevents double spending	Punishment for validating nodes	Suitable for asynchronous networks
			Nodes are known	Nodes are known	Nodes are known		Addresses scalability issue concerning network size and transaction throughput

Figure D.1: Comparison of Consensus Protocols

Appendix E

Transactions per Second for Selected Cryptocurrencies

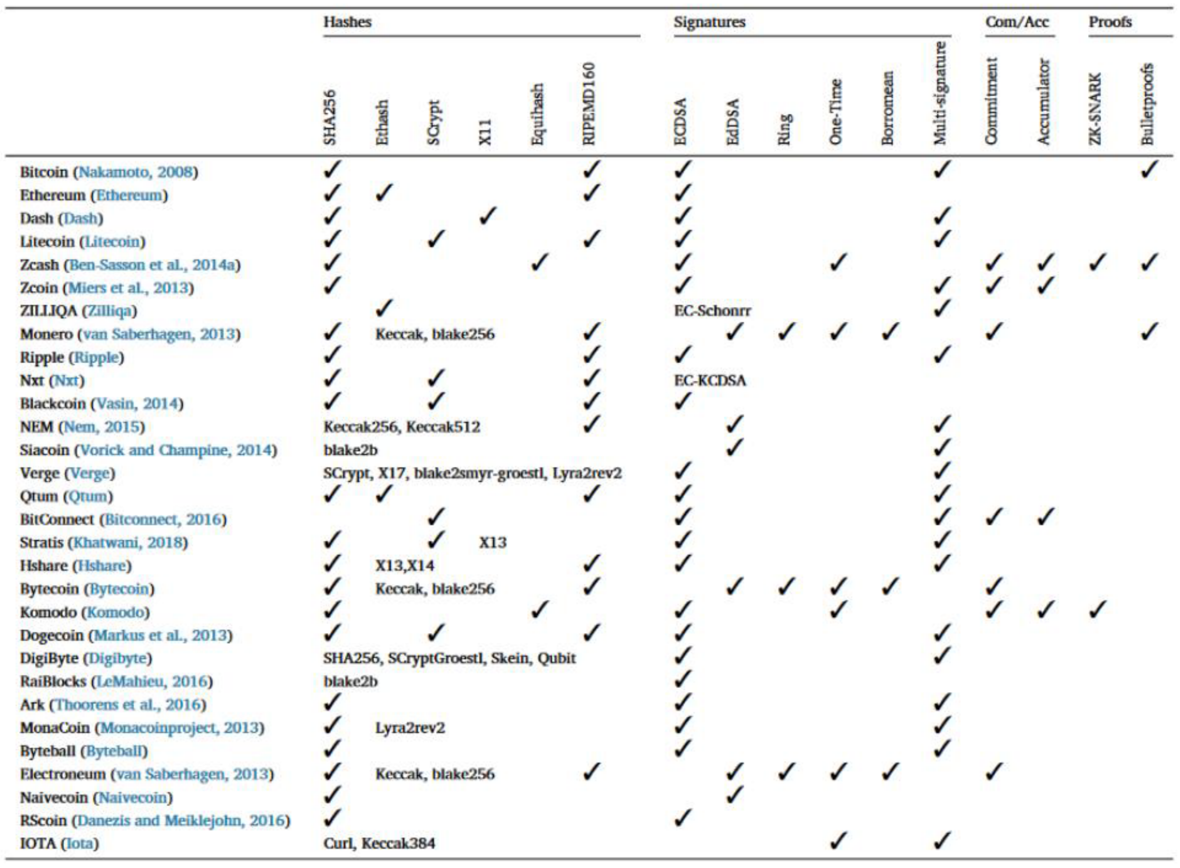


Cryptocurrency Name	Protocol	TPS
Bitcoin	PoW	7
Ethereum	PoW	15
Ripple	RPCA	1500
Bitcoin Cash	PoW	60
Cardano	PoS	7
Stellar	SCP	1000
NEO	DBFT	10000
Litecoin	PoW	56
EOS	DPoS	~millions
NEM	PoI	4000

Figure E.1: Transactions per Second for Selected Cryptocurrencies

Appendix F

Cryptographic primitives in blockchains



	Hashes						Signatures						Com/Acc		Proofs	
	SHA256	Ethash	SCrypt	X11	Equihash	RIPEMD160	ECDSA	EdDSA	Ring	One-Time	Borromean	Multi-signature	Commitment	Accumulator	ZK-SNARK	Bulletproofs
Bitcoin (Nakamoto, 2008)	✓					✓	✓					✓				✓
Ethereum (Ethereum)	✓	✓				✓	✓									
Dash (Dash)	✓			✓			✓					✓				
Litecoin (Litecoin)	✓		✓			✓	✓					✓				
Zcash (Ben-Sasson et al., 2014a)	✓				✓		✓			✓			✓	✓	✓	✓
Zcoin (Miers et al., 2013)	✓						✓					✓	✓	✓		
ZILLIQA (Zilliqa)		✓					EC-Schonrr					✓				
Monero (van Saberhagen, 2013)	✓	Keccak, blake256				✓		✓	✓	✓	✓		✓			✓
Ripple (Ripple)	✓					✓	✓					✓				
Nxt (Nxt)	✓		✓			✓	EC-KCDSA									
Blackcoin (Vasin, 2014)	✓		✓			✓	✓									
NEM (Nem, 2015)	Keccak256, Keccak512					✓		✓				✓				
Siacoin (Vorick and Champine, 2014)	blake2b							✓				✓				
Verge (Verge)	SCrypt, X17, blake2smyr-groestl, Lyra2rev2						✓					✓				
Qtum (Qtum)	✓	✓				✓	✓					✓				
BitConnect (Bitconnect, 2016)			✓				✓					✓	✓	✓		
Stratis (Khatwani, 2018)	✓		✓	X13			✓					✓				
Hshare (Hshare)	✓	X13,X14				✓	✓					✓				
Bytecoin (Bytecoin)	✓	Keccak, blake256				✓		✓	✓	✓	✓		✓			
Komodo (Komodo)	✓				✓		✓			✓			✓	✓	✓	
Dogecoin (Markus et al., 2013)	✓		✓			✓	✓					✓				
DigiByte (Digibyte)	SHA256, SCryptGroestl, Skein, Qubit						✓					✓				
RaiBlocks (LeMahieu, 2016)	blake2b						✓									
Ark (Thoorens et al., 2016)	✓						✓					✓				
MonaCoin (Monacoinproject, 2013)	✓	Lyra2rev2					✓					✓				
Byteball (Byteball)	✓						✓					✓				
Electroneum (van Saberhagen, 2013)	✓	Keccak, blake256				✓		✓	✓	✓	✓		✓			
Naivecoin (Naivecoin)	✓							✓								
RScoin (Danezis and Meiklejohn, 2016)	✓						✓									
IOTA (Iota)	Curl, Keccak384									✓		✓				

Figure F.1: Cryptographic primitives in blockchains