Tampere University

Juho Yrjölä

FIPA-COMPLIANT MICROPAYMENTS USING BITCOIN LIGHTNING NETWORK

Master of Science Thesis
Faculty of Engineering and Natural Sciences
Examiners: Professor Jose Martinez Lastra
University Instructor Luis Gonzalez Moctezuma
February 2022

ABSTRACT

Juho Yrjölä: FIPA-compliant micropayments using Bitcoin Lightning Network
Master of Science Thesis
Tampere University
Master's Programme in Automation Engineering
February 2022

Due to the growth of popularity in the fields of Internet of Things (IoT) and automation, multitude of use cases for Machine to Machine (M2M) payments is expected emerge. Some of these use cases require payments of very small value, called micropayments. In this thesis a proof-of-concept protocol for sending M2M micropayments is designed and implemented. The technologies that were chosen for the implementation are JADE-framework that is based on FIPA multiagent-system, and Bitcoin Lightning Network (LN).

The objective of the design and implementation of the system is to find out if it is possible to create this type of protocol to be affordable and fast enough to perform well in multitude of micropayment use cases. The protocol is designed in a way that the counterparties of a payment can denominate the value of the payment in a traditional currency such as euro. Also, the design and implementation are done in a way that they are easily adaptable to multitude of use cases.

The system is based on two types of JADE-agents: a payment sender agent and a payment receiver agent. The agents are linked to their respective LN nodes, via which the payment is conveyed. LN node is a software capable of storing, sending and receiving bitcoin on the LN. To denominate the value of the payment in traditional currency, each agent fetches the price of bitcoin independently from an external web resource.

A series of payments was made using the implementation to measure the speed and affordability of the payments. The fee for each payment was around 0.0005 euro, or 0.05 cents. The time to complete a payment using the protocol varied between 3 and 30 seconds. The time was longer than expected and was assumed to be mostly caused by the constrained hardware that was used to run the LN nodes. It was concluded that a similar protocol to the one that was designed could be a suitable infrastructure for M2M micropayments in the future.

Keywords: bitcoin, lightning network, agent, FIPA, multi-agent system, micropayment, microtransaction

TIIVISTELMÄ

Juho Yrjölä: FIPA-yhteensopivat mikromaksut Bitcoin-salamaverkossa
Diplomityö
Tampereen yliopisto
Automaatiotekniikan diplomi-insinöörin tutkinto-ohjelma
Helmikuu 2022

Esineiden internetin (IoT) suosion ja automaatioasteen kasvun myötä koneiden välisille (M2M) maksuille odotetaan ilmaantuvan lukuisia mahdollisia käyttökohteita. Jotkut näistä käyttökohteista edellyttävät toistuvia, erittäin pieniarvoisia maksuja, joita kutsutaan mikromaksuiksi. Tässä diplomityössä suunnitellaan ja toteutetaan prototyyppi-protokolla M2M-mikromaksujen lähettämiseen. Toteutukseksi valittiin FIPA-moniagenttijärjestelmään perustuva JADE-ympäristö ja Bitcoinin salamaverkko (Lightning Network, LN).

Järjestelmän suunnittelun ja toteutuksen tavoitteena on selvittää, onko mahdollista luoda tällainen protokolla niin edulliseksi ja nopeaksi, että sitä voitaisiin käyttää useissa mikromaksujen käyttökohteissa. Protokolla on suunniteltu siten, että maksujen vastapuolet voivat määrittää maksun arvon perinteisessä valuutassa, kuten euroissa. Suunnittelu ja toteutus pyritään tekemään niin, että järjestelmä on helposti mukautettavissa moniin käyttökohteisiin.

Järjestelmä perustuu kahden tyyppisiin JADE-agentteihin: maksun lähettäjäagenttiin ja maksun vastaanottaja-agenttiin. Agentit on linkitetty salamaverkkosolmuihinsa (LN node), joiden kautta maksu välitetään. Salamaverkkosolmu on ohjelmisto, jolla voidaan säilöä, lähettää ja vastaanotaa bitcoineja salamaverkossa. Maksun arvon määrittämiseksi perinteisessä valuutassa kumpikin agentti noutaa bitcoinin ajantasaisen hinnan itsenäisesti ulkoisesta verkkoresurssista.

Toteutusta hyödyntäen tehtiin sarja maksuja, jotta voitiin mitata maksujen nopeutta ja kulujen määrää. Kulu kustakin maksusta oli noin 0,0005 euroa eli 0,05 senttiä. Maksun suorittamiseen kulunut aika protokollaa käyttäen vaihteli 3 ja 30 sekunnin välillä. Maksujen nopeus oli odotettua hitaampi. Hitauden oletettiin johtuvan pääosin pienitehoisesta minitietokoneesta, jota käytettiin salamaverkkosolmujen pyörittämiseen. Diplomityön loppupäätelmä on, että suunnitellun kaltainen protokolla voisi tulevaisuudessa toimia koneiden välisten mikromaksujen toteuttavana infrastruktuurina.

Avainsanat: bitcoin, lightning network, salamaverkko, agentti, FIPA, mikromaksu, mikrotransaktio

PREFACE

The initial motivation for this thesis was my conviction of bitcoin being a viable and better alternative for the current global monetary system. Therefore, I want to thank Satoshi Nakamoto for inventing (or discovering) the technology and thus providing the fundamental basis for this thesis.

I also want to thank my examiners, prof. Jose Martinez Lastra and university instructor Luis Gonzalez Moctezuma, for supporting me in the process. I further want to thank my girlfriend Mari for support and motivation.

Tampere, 22nd February 2022

Juho Yrjölä

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF PROGRAMS AND ALGORITHMS

LIST OF SYMBOLS AND ABBREVIATIONS

ACL	Agent Communication Language
AIP	Agent Interaction Protocol
API	Application Programming Interface
AUML	Agent UML
Bitcoin	The peer to peer network or technology that is enabling the digital currency
bitcoin	The digital currency
BOLT	Basis of Lightning Technology
CA	Communicative act
DAG	Directed Acyclic Graph
DLC	Discreet Log Contract
DLT	Distributed Ledger Technology
EAV	Electric Autonomous Vehicle
FIPA	Foundation for Intelligent Physical Agents
gRPC	gRPC Remote Procedure Call
HTLC	Hashed Timelock Contract
IoT	Internet of Things
IP address	Internet Protocol address
JADE	Java Agent Development Environment
JSON	JavaScript Object Notation
LAN	Local area network
LN	Lightning Network
LND	Lightning Network Daemon
M2M	Machine to Machine
MAS	Multi-Agent System
mempool	A list of pending transactions that are to be included into a blockchain
micropayment	A monetary payment of small value

MQTT	Message Queuing Telemetry Transport
PCN	payment channel network
PoW	Proof-of-work
REST	Representational State Transfer
RPC	Remote Procedure Call
satoshi	The smallest unit of bitcoin, often abbreviated to sat
SHA-256	A hashing algorithm used by the bitcoin network
SOLID	A collection of five priciples that are considered best practices in object-oriented programming
TCP	Transmission Control Protocol
UML	Unified Modeling Language

1. INTRODUCTION

1.1 Background

Current advancements in Internet of Things (IoT) and Machine to Machine (M2M) interactions have introduced possible use cases for M2M micropayments [1][2]. One of the possible use cases is trading data that could originate from variety of sensors or from processed datasets [1][3]. Another example of M2M micropayments could be paying for the charging of the batteries of electric autonomous vehicles [4]. As devices get smarter and the level of automation increases throughout economies, more and more use cases could emerge.

Micropayments are payments of small monetary value [1]. *A dictionary of the Internet (2019)* [5] defines a micropayment as a monetary transaction of such a low value that it is not economically feasible to process it in a traditional way like using a bank or other intermediary. There is no universal definition as to what the absolute maximum value is to be deemed as a micropayment. For example, PayPal considers micropayment to be a payment that is typically lower than 10 U.S. dollars [6], but this does not coincide with the aforementioned definition. In this thesis values ranging from approximately one euro cent (0.01 €) to one euro (1.00 €) are recognized as micropayments.

Traditional payment models require a trusted third party such as a bank or other intermediary to deliver a payment. For M2M micropayments this is not an optimal model for many reasons. For many use cases it is a not desirable from a security point of view to connect a bank account to the large amount of devices that might be employed. Also, there can be some supplementary effort in negotiating contracts with the intermediary for the integration of the payment system. More importantly, as covered by the prior definition, the intermediaries charge fees that are often too high compared to the value of the transactions. Also, the transaction times might be too slow, since the payment has to be processed in the third-party system. [7][8][9][10]

One way to solve the problem of the fees being too high relative to the size of the transaction is by using blockchain or blockchain based technology. Blockchain was first introduced by a pseudonymous person or group of persons called Satoshi Nakamoto in their paper which describes Bitcoin [11]. Blockchain is a distributed ledger where transactions

of monetary value are stored. The advantage of using blockchain in a transfer of monetary value is that the payments do not need to be processed by a central authority. Instead, the payments are processed by a decentralized peer to peer network. In the case of a permissionless blockchain such as Bitcoin there are no limitations as to who can join the network [12]. This model which circumvents the need for a central authority might be suitable for and M2M micropayments by lowering transaction fees and processing times compared to traditional models.

In this thesis Bitcoin is chosen as the blockchain to be utilized for the micropayments, because it is the first and most established of all the public blockchains, thus having high credibility of being a reliable and dependable system [13]. Bitcoin is a system that is governed by the consensus of the network participants. Since the inception of Bitcoin, the fundamental properties of it have never been altered [14][13], and therefore it can be thought to be practically unchangeable. However, there have been credible attempts to make changes to the protocol [15][13], but as the value and relevance of Bitcoin increases over time it is less and less likely for any attempt to succeed. In other notable public blockchain systems such as Ethereum, the fundamental properties called the consensus rules change from time to time [16], so the guarantee of long-term stability of the system is inferior to Bitcoin. This is one key aspect of Bitcoin that fundamentally differentiates it from most other public blockchain systems [13]. The Section 1.2 further examines the relevance and emergence of Bitcoin in the global economy.

The Bitcoin base protocol itself is not suitable for micropayments, because of slow transaction times and relatively high fees [1]. Blocks on Bitcoin blockchain emerge on average every 10 minutes, which creates a restriction on the speed of transactions on Bitcoin [11]. The fees vary a lot over time. The fees have historically ranged from practically negligible amounts to as high as over 50 U.S. dollars per transaction. At the time of writing the fees have ranged from 0.1 to 5 U.S. dollars per transaction [17]. Therefore, the fee for Bitcoin cannot be trusted to be low enough for micropayments. A second layer solution called Lightning Network (LN) [18] is a technology that is developed on top of Bitcoin base protocol to overcome these problems of slow transaction times and high fees. LN uses a network of payment channels that keep track of Bitcoin transactions between participants without broadcasting them to the Bitcoin blockchain. The designers of LN propose that the network enables near-instant transactions with negligible fees [18].

One approach for M2M communication is to use a Multi-Agent System (MAS). MAS is a software architecture that is based on individual software entities called agents, that communicate with each other acting with varying degree of autonomy to reach their goal. MAS was chosen as the base framework, since it is a good representation of a system where individual software components interact in a peer to peer fashion. In MAS, for an M2M scenario one machine could correspond to one agent. In this thesis a popular MAS-framework defined by Foundation for Intelligent Physical Agents (FIPA) is used as

the framework for M2M interactions. The objective of the thesis is not to employ M2M micropayments for a specific use case, but instead design a general solution for M2M micropayments. The desired result of the thesis is to present a FIPA-compliant micropayment solution, that could be employed for multitude of M2M micropayment use cases.

1.2 On the relevance of Bitcoin

Bitcoin is an emerging technology that allows individuals, corporations and even governments to store and transfer monetary value. The fundamental value proposition of Bitcoin is that the value stored and transacted in the network is not controlled by any single entity, but by the decentralized network [11]. Another important aspect of Bitcoin is the limited amount of bitcoin that will ever exist [13]. This establishes the first widely adopted occurrence of absolute scarcity of a money supply [19]. In fact, in his book *The Bitcoin Standard* [20] Saifedean Ammous, a PhD in Sustainable Development, argues that bitcoin is the first commodity to ever exist that has a strictly limited supply [20, Chapter 9.]. Some of the most radical proponents of Bitcoin consider the implications of these features to be of such a significance that they perceive a possibility of bitcoin becoming the new world reserve currency sometime in the future [20, Chapter 9.] [21] [22][23]. This situation is often referred as The Bitcoin Standard. It could either mean that bitcoin is the sole currency that is used in everyday transactions by regular individuals, or that nations would issue national currencies whose exchange rate is tied to bitcoin which is used to back these currencies. In fact, thus far there is already one nation; El Salvador, that has taken the first step towards the Bitcoin Standard by purchasing a considerable amount of bitcoin on the balance sheet of the country, as well as adopting bitcoin as a legal tender besides the U.S. dollar [24][25, p. 50]. It remains to be seen if there will ever be other nations to follow their example or if El Salvador will act as a failed experiment.

Bitcoin also has multiple critics who have various concerns on why it is inconceivable for bitcoin to become widely adopted money. One of the significant arguments against wide adoption of Bitcoin is that governments and central banks would not allow a competing money to emerge [26]. Another common criticism is the implications of the deflationary nature of bitcoin as money [27][28]. Critics argue that as the amount of bitcoin is limited, the value would grow over time, which incentivizes hoarding of bitcoin and therefore not spending or investing it. This would in turn lead to stagnation or even shrinking of the economy as economic activity declines.

The current significance of Bitcoin in the global economy can be quantified with multiple numeric metrics. A few of these are market capitalization and annual transfer volume. Market capitalization of bitcoin is the total market value of all bitcoin currently in existence. At the time of writing, the market capitalization is currently around 700 billion (7×10^{11}) U.S. dollars, and at one point during the year 2021 it was as high as over 1.2 trillion

Figure 1.1. *The price history of bitcoin since the beginning of 2013 [29].*

(1.2×10^{12}) U.S. dollars [30]. This can be compared to the total value of all the gold that has ever been mined, which is estimated to be around 11.4 trillion (1.14×10^{13}) U.S. dollars [31]. Therefore, the value of all bitcoin is in the magnitude of around 10 % of the total value of all gold. The value of bitcoin has grown to this proportion in the 13 years of its existence. The historical value growth of a single bitcoin is shown in Figure 1.1. The y-axis, which represents the price of a single bitcoin in U.S. dollars, is presented on logarithmic scale since the price values span multiple decades. Another way to quantify the significance of Bitcoin is to estimate the amount of value that has been transacted in the network. The total value transacted in the Bitcoin network during the year 2021 was 13.1 trillion (1.31×10^{13}) U.S. dollars, which is around 20 % higher that the annual payments volume of the financial services company Visa [25, p. 47]. By this metric it can be concluded that Bitcoin is already a significant payment network in the global scale.

1.3 Objectives and research questions

The objective of this thesis is to design, implement, test and analyse a micropayment protocol using Bitcoin LN that is fully compliant with the FIPA-MAS framework. For testing and analysing the implementation, the following research questions are posed to further clarify the scope of the thesis:

- Is it possible to create a FIPA-compliant payment protocol based on Bitcoin LN?
- Is Bitcoin Lightning Network a valid technology to be used for agent to agent micropayments?
  - Is it possible to consistently achieve transaction time less than 1 s and a fee less than 0.01 euro?
- In what conditions or circumstances would agent to agent or machine to machine transactions using Lightning Network be usable for a business?

Even though the software is intended to be designed in a way that it could be adapted to a wide variety of use cases, the design and development is purely research-driven. Therefore, the arising software is not intended to be directly used in a production environment. Because of this, only functionalities relevant to the research questions need to be implemented. To narrow down the scope of the implementation, some functional and non-functional requirements for the implementation are posed in Section 3.1.

1.4 Outline

The chapter 1 introduces the problem and the research questions that act as the basis for this thesis. The chapter 2 introduces the technologies that are used in the thesis, such as MAS, Bitcoin and LN. Relevant earlier research on the topic of micropayments using blockchain technology is also reviewed. Chapter 3 outlines the conceptual design of the system that is developed for this thesis. This includes defining the components of the system and the communication patterns that will be used. The implementation of the system is described in Chapter 4 as well as employing the system to a specific test scenario. In Chapter 5 the performance and applicability of the system is tested, analysed and discussed. Chapter 6 concludes the thesis and provides topics for further research.

2. BACKGROUND ON THE UTILIZED TECHNOLOGIES AND RELATED EXISTING IMPLEMENTATIONS

This chapter provides technical background for the thesis. In first three parts of the chapter, the technologies that form the base for the research are presented. In the last part some relevant prior research on the topic is outlined.

2.1 FIPA Multi-agent systems

MAS is a system that is comprised of autonomous programmatic entities called *agents*. The agents of a MAS must satisfy some conditions [32] to be considered agents in the sense that is implied in this thesis. An agent must have some level of *autonomy* such that it can act without being directly operated by humans. An agent must have *social ability*, meaning that it is able to communicate with other agents using some Agent Communication Language (ACL). It must also be *reactive*, meaning that it can perceive and react to the environment it is in. Finally, an agent must have the capability to be *pro-active* such that it can initiate actions and not just react to it's environment. Agents communicate with one another in order to fulfill the purpose that the system of agents is designed for. The mode of communication, level of autonomy of the agents as well as other properties of the agents are determined by the design and the use case of the system. One essential characterization of a MAS is on what basis do the agents choose to act and communicate. The internal motivations for agents might for example arise from goals that are given to them. MAS as a computational architecture is well suited in situations which naturally consist of multiple entities that operate in a dynamic environment where the amount and types of tasks are not constant. [33]

FIPA is an organization that has standardized one major MAS framework [34]. The mission of FIPA is to promote the industry of intelligent agents. Although, the organization is not active anymore, a survey from 2018 [35] claims that the framework defined by FIPA is still the most widely adopted model for MAS. A FIPA-compliant MAS is a system that satisfies the specifications devised by FIPA. The specifications mostly define the interactions between agents rather than the internal behaviour of the agents, since the internal behaviour is largely concealed [33]. The FIPA specifications define an ACL called FIPA-ACL, which is the agent interaction model used in FIPA-compliant systems. FIPA-ACL is in

fact much more than just a communication language. FIPA-ACL defines multiple different components that facilitate standardized way of communication between agents. The core of FIPA-ACL are the Communicative acts (CAs), that are standardized message types that can be sent among agents. The choice of CA defines the context of a message. For example, some CAs determine that a message is a question or a request, while others define that the intent of a message is providing information. Another important component of FIPA-ACL are Agent Interaction Protocols (AIPs). An AIP defines a communication pattern between agents, by specifying a sequence of CAs between agents. Each AIP is intended for a specific type of interaction. FIPA-ACL also contains a concept of ontology. It is a set of concepts, meanings and semantics used in a specific situation or domain. It also defines the relationships between the concepts. Each FIPA-compliant implementation can define a distinct ontology, so that the agents have a common agreement of the domain that they operate in and can communicate without misunderstandings. [33]

Agent interactions are often presented using Agent UML (AUML), which is an extension of Unified Modeling Language (UML) [36]. Notably, diagrams that are based on the UML sequence diagram are often used to present the AIPs. In these diagrams the UML object is replaced by an agent and an arrowed line is represents a CA instead of an object oriented style message. This way an AIP can be explicitly and clearly presented, and it is easy to interpret the progress of “discussion” between the agents.

2.2 Bitcoin

Bitcoin is a complex peer to peer network, designed to transfer and hold value without a trusted third party [11]. This chapter aims to explain the general principles on how the Bitcoin network operates. The Lightning Network uses specifically the ability of Bitcoin to conduct conditional payments that are called smart contracts [18]. This is why the Bitcoin transaction scripting that enables the smart contracts is emphasized in this chapter.

There are multiple problems in creating a peer to peer monetary system that does not require a trusted third party. Notably the double-spend problem was not properly solved until the pseudonymous Satoshi Nakamoto proposed a solution in the original Bitcoin paper [11]. For money to have value, it is important that the owner of the money cannot spend the same money multiple times, since otherwise they would practically have an infinite amount of money. As it is trivial to duplicate any digital data, there must exist a mechanism to prevent the duplication of the digital object that contains the value.

To solve the double-spend problem Bitcoin uses a distributed ledger where all the participants have the same copy of the ledger and thus agree on the state of the system. A participant in the network is called a *node*. The ledger is organized in a chain of blocks. Each block contains the hash value of the earlier block which guarantees that the blocks are ordered chronologically. This construct is nowadays called the blockchain. Figure 2.1

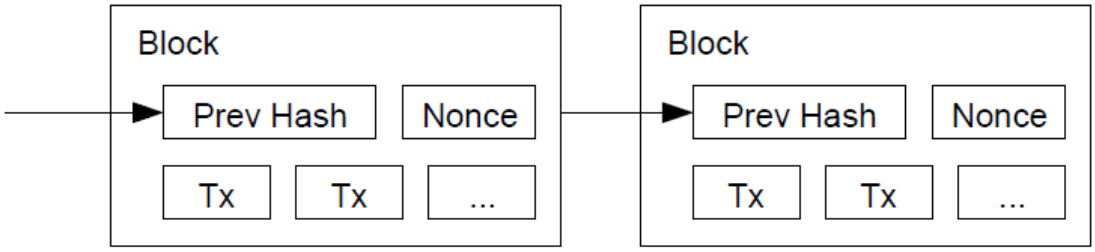

Figure 2.1. *Two blocks on the blockchain [11].*

presents a visualization of two chained blocks on the blockchain. The payload of a block in the blockchain is the transactions between the nodes of the network. The size of each block is limited, which limits the amount of transactions that can be included in a block. [11]

To reach a consensus between the nodes on which blocks to add to the blockchain, a mechanism called Proof-of-work (PoW) is used. To post a new block to the blockchain, a node must provide a proof that they have spent a certain amount of computational resources. This is implemented so that the SHA-256 hash value of the new block has to begin with a certain number of zero-bits. The SHA-256 hash is seemingly random and it is impossible to affect the outcome of the hash [37]. Therefore, the only means of reaching a valid hash value is by randomly hashing blocks with slightly different contents, until a block that satisfies the requirement is found. When a node that is trying to generate a new block announces a block with valid contents and a valid hash, the other nodes accept it to be added to the blockchain. A node which is trying to generate new blocks is nowadays called a miner. To incentivize miners to generate new blocks, a reward in the form of new bitcoins is given to the one which succeeds in creating a new block. This is also the way in which all the bitcoins are initially created. Over time the block reward decreases asymptotically towards zero. This ensures that there is a finite amount of bitcoins that can ever exist. [11]

The PoW is essential to guarantee that no malicious actor can harmfully influence the contents of new blocks without spending significant amount of costly resources. PoW also limits the speed of generating new blocks, which in turn limits the amount of disk space needed to store the blockchain and allows all the nodes to keep up to date on the state of the blockchain [11]. The protocol guarantees that over time the average time between new blocks is ten minutes [38]. Most of the nodes on the network are not miners. It is very cheap to run a non-miner node, since the only major computation needed is verifying the has of each new block. Even a cheap single-board computer such as Raspberry Pi with an external hard-drive is powerful enough to run a Bitcoin node [39, Chapter 3.]. In 2019 there were around 10000 geographically distributed nodes that participate in verifying the validity of new blocks [40]. The large number of nodes reinforces the security of the network, as there would need to be a significant amount of colluding malicious nodes to

undermine the security of the network.

A transaction in Bitcoin has at least one input and at least one output. A transaction always spends the output of an earlier transaction. Technically there are no wallets or addresses in Bitcoin. A transaction output can be spent by anyone who can satisfy the requirements set by the earlier transactor. The requirements are set using the Bitcoin scripting language. The scripting language consists of a set of commands that are executed in sequence. The language does not include loops to make the execution more predictable, and to avoid the possibility of getting stuck in the execution. A *standard* Bitcoin transaction makes use of the digital signature of public-key cryptography. A standard transaction has essentially a requirement such that: the spender must provide the digital signature for the public key that the earlier transactor has set as the receiver of the transaction. Some other examples of transactions that can be constructed include a transaction where funds can be spent only after a certain time has passed and a transaction where multiple signatures of different public keys are required. In fact, these two types of transactions called *timelock* and *multisig* transactions are used to enable the Lightning Network [18]. A non-standard Bitcoin transaction is sometimes called a smart contract. The capabilities of smart contracts on Bitcoin are very limited in comparison to some other blockchains that facilitate the execution of arbitrarily complex smart contracts. [41] [42]

Any node can announce a transaction to other nodes, who then propagate it to their peers. This way the miners receive the transactions that they can add to a block. The list of pending transactions that are announced, but not yet included in the blockchain is called the memory pool or *mempool*. To incentivize the miners to include a transaction in the block, the creator of the transaction can include an additional reward that they pay the miner in case the transaction is included. [43] [44]

The specification of Bitcoin is defined by the reference implementation called Bitcoin Core which is based on the original implementation created by Satoshi Nakamoto [45]. The software is developed in an open source manner and changes to the software are thoroughly reviewed [46]. Even after a change to the software is accepted, it is up to the nodes to decide whether to accept the change by updating their node to the new version. Therefore, it is eventually the consensus of the nodes that decide what is Bitcoin. The overwhelming majority of Bitcoin nodes run some version of the Bitcoin Core. According to bitnodes.io at the time of writing, over 99 % of all the nodes on the Bitcoin network run some version of the Bitcoin Core software [47].

2.3 Lightning Network

Lightning Network (LN) is a layer 2 technology that is leveraging the programmable properties of Bitcoin transactions to create transactions that are not included in the blockchain.

This chapter explains the operation of the LN and the constraints it has. Since the transactions on the lightning network are not posted on the blockchain, the limitations on capacity and time do not apply. Because of this, LN can potentially enable nearly unlimited amount of practically instantaneous and cost-effective transactions [18]. Unlike Bitcoin, LN is defined by a set of written specifications called Basis of Lightning Technology (BOLT) [48, Chapter 4.] [49]. LN consists of LN nodes that are connected to each other, forming a network. A LN node is a software program that realizes the BOLT specifications. A LN node must contain or have a connection to a Bitcoin node to access the blockchain. There are three major interoperable LN node implementations which are all open source projects and are developed by three separate organizations [48, Chapter 4.].

Lightning Network is a network of payment channels. A payment channel is a separate ledger formed between two LN nodes. A crucial part of forming a payment channel is posting a specific multisig transaction on the Bitcoin blockchain. To spend the multisig transaction it requires the signature of both nodes between whom the transaction was created. This initial transaction is called the *funding transaction*. The funding transaction includes a transfer of funds from one or both of the parties. The funds included in the funding transaction are the funds that are usable in the LN. The funds are locked in the LN until another transaction is posted on the blockchain to close the payment channel. After the channel is closed the funds become standard spendable outputs on the Bitcoin blockchain. [18]

At the time of creating the funding transaction, the nodes also create transactions called *commitment transactions*. They do not post the commitment transactions on the blockchain, but instead, at this moment, they store the commitment transactions locally. In the future they can post the commitment transaction to the blockchain if they want to close the channel for any reason. Each of the nodes store a commitment transaction signed by the other node, so either of the nodes can close the channel without requiring further permission from the other node. When a transaction is made on the LN, new commitment transactions are created that overwrite the earlier transactions. The latest commitment transaction holds the current balances between the participants of the channel. Creating new commitment transactions therefore update the balances of funds belonging to each of the nodes. As the commitment transactions are not posted on the blockchain while the channel is open, it is possible to do a very large amount of transactions very cheaply. In fact if the nodes agree not to charge any fees, the transactions are feeless. One major constraint on the LN payments is the channel capacity. Making a transaction is done by updating the channel balances. However, if the state of the channel is such that all the funds of the channel already belong to either of the nodes, it is not possible for the other node to make a transaction since they do not have anything to send. Likewise, if the amount to be transacted is higher than the available balance, it is not possible to make the transaction [48, Chapter 3.]. The amount of funds on a LN channel that a node can

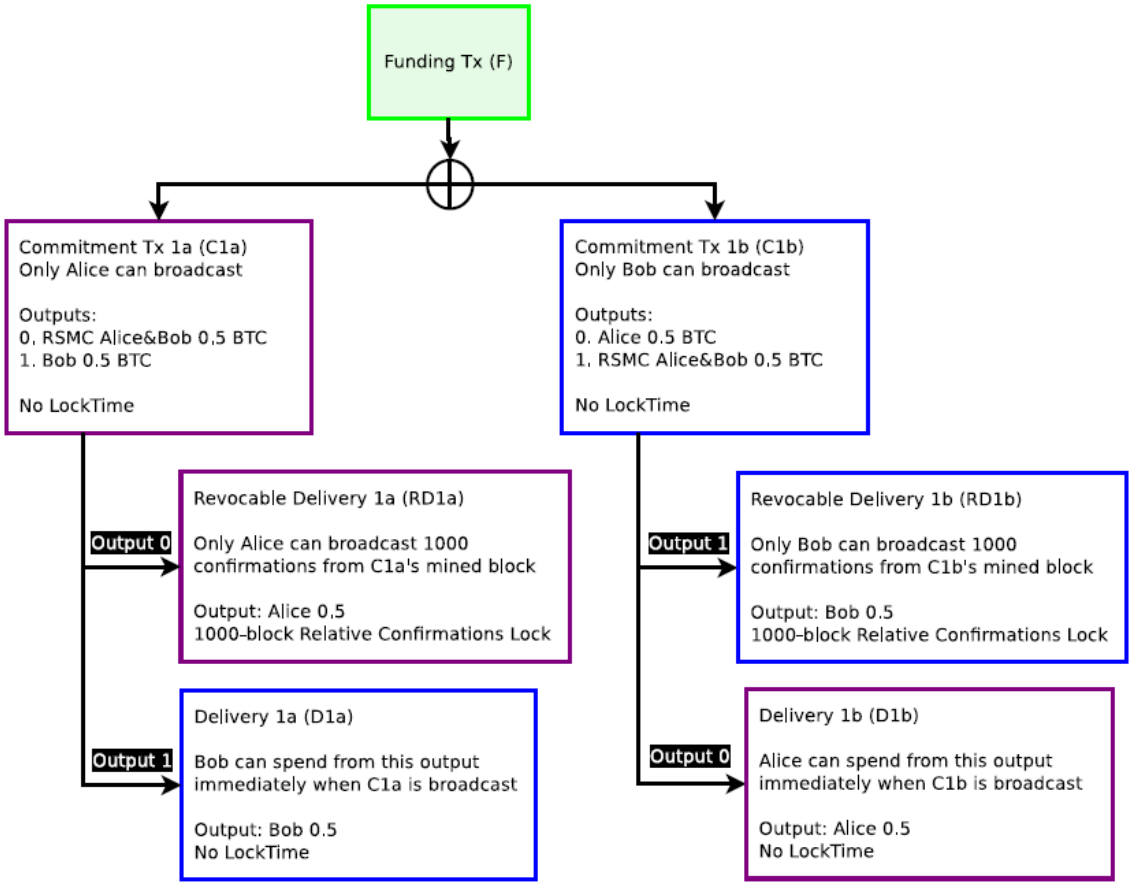

Figure 2.2. *The commitment transactions spend the output of the funding transaction. Blue and purple colors indicate which party can broadcast a transaction on the blockchain. [18]*

still receive is called inbound capacity or inbound liquidity [48, Chapter 5.]. Respectively, the amount available for sending is called outbound capacity or outbound liquidity.

The structure of the commitment transactions is presented in Figure 2.2. The old commitment transactions remain valid transactions that can be posted to the blockchain. This would enable stealing of funds, as a node could redeem an earlier balance that is higher than that of the latest commitment transaction. To ensure that old commitment transactions are invalidated, the commitment transactions involve means to revoke the earlier ones. The corresponding commitment transactions that each node have, are not actually exact duplicates of each other. They both accredit the latest agreed balances to each node, but they favour the counterparty. The corresponding commitment transactions are created such that when posted to the blockchain, the counterparty can spend the funds belonging to them immediately. On the other hand, for the one who posted the transaction there is a timelock in place such that they must wait a certain time to access their funds. To create a new commitment transaction that revokes the earlier one, each node gives a private key to the other node that allows them to access funds of the counterparty, if the revoked transaction is posted to the blockchain. When a revoked commitment transaction

is posted, the wait time of the timelock starts after which the one posting the transaction will get the access to the funds that belonged to them at the time. Since the transaction is outdated and was revoked, the other party now has the private key to redeem the time-locked funds along with the funds that belonged to them anyways. If they use the private key to access the funds before the timelock runs out, they immediately get the funds that would have been accessible to the one posting the outdated transaction after the time-lock expires. The software is built so that the private key is used immediately when the other party tries to cheat, by posting a revoked commitment transaction. This structure enforces the current balances and makes sure that neither of the channel participants can steal the funds of the other by trying to redeem outdated balances. In fact, a proper software implementation should automatically delete the revoked commitment transactions, to make sure that they are never posted. If both parties are cooperative as usually is the case, they can agree to close the channel by creating a transaction with current balances without any timelocks. [18]

The Lightning Network is a network that is constructed of payment channels described above. To send a payment in the LN, the sending and receiving nodes do not need to have direct payment channel between them, but the payment can be routed through multiple connected payment channels. Payment channels are connected by nodes that have multiple payment channels connected to them. The payment can be made as long as there is a route of payment channels with sufficient capacity and the nodes along the route are willing to route the payment. Often the payment route passes through a well connected node called a hub [50]. There is no imposed limit on how many payment channels a node can have open. The amount of channels per node is limited by the funds that the node is willing to commit to the LN. Opening and maintaining payment channels is incentivized by routing fees; a node can impose a fee for every payment that is routed through it [51]. A dataset collected in April 2019 suggests that 87 % of the nodes can find a route of 8 steps or less to any other node and 90 % of nodes can find a route to 90 % of the nodes in 4 steps [52].

The multi-hop payments passing through intermediary nodes also respect the principle of trustlessness. Moreover, as on the simple payment channels, if all the participants are co-operative, there is no need to post any transactions to the blockchain. Again, commitment transactions are used as an assurance to guarantee that the arrangement is enforceable. Once a route for a payment is found, where all the intermediary nodes tentatively agree to route the payment, the original sender and each node in the route generate a new commitment transaction that, under certain conditions, sends a payment to the next node in the route. These commitment transactions include an additional construct called Hashed Timelock Contract (HTLC). HTLC is an additional output for the commitment transaction that contains the funds to be routed. HTLC is conditional transaction that can send the funds either to the next node in the route or back to the intermediary node who created

the HTLC. If the next node can prove that they know a secret called a *preimage*, the funds are instantly sent to them in the case that the commitment transaction is posted on the blockchain. However, if the preimage is not provided, there is a timelock after which the funds are sent back to the sender. The final receiver generates the preimage and provides the hash of it to the routing nodes, for them to construct the HTLCs. Once all of the HTLCs are created, the final receiver reveals the preimage to last node in the route. The preimage is then sequentially revealed to all of the nodes in the route after which all of the nodes in the route agree that any of the nodes can post their HTLC to the blockchain to claim their funds. At this point, if all the nodes are cooperative, they create new commitment transactions to update each payment channel to the state where the routed payment is made and revoke the HTLC. [18]

A payment in the LN is usually initiated using a *payment request*, also called an *invoice*. The format of the invoice is defined in BOLT-11 [53]. The invoice is created by the receiver of the upcoming payment and relayed to the payer outside of LN. The invoice can be delivered in via any medium that supports the transmission of textual data. The invoice contains the hash of the preimage, the address of the receiver and a time of expiry. The hash of the preimage is also called the *payment hash* and it is the unique identifier used to identify payments. It is optional to include the amount to be sent, a textual description of the payment and hints on how to route the payment. A separate BOLT-11 invoice must be generated for each payment, so it is not practical for recurring payments. [48, Chapter 3.] A draft specification of BOLT-12 is proposed as a more versatile payment request called an *offer* [54][55]. It would allow non-expiring recurring payments and payments denominated in currencies other than bitcoin.

There are some disadvantages to the LN. One of the most prominent problems is the problem of routing channel balances. If enough payments are sent via the same route, all the balance on that route is accumulated on one side of the channel, making it impossible to send anymore payments to that direction until the channels are rebalanced [2]. This is a well known problem and multiple parties are trying find optimizations to alleviate this [2][51]. There are also other, less fundamental problems. To enforce their payment channel balances, a node must stay online monitoring the blockchain to be able to react if the counterparty decides to post a revoked commitment transaction that does not represent the current balances. This action is done automatically by the software, but if the software is terminated or taken offline, the counterparty has the possibility to cheat. A less serious complication of going offline is that the counterparty might close the payment channel, if they think that the payment channel is obsolete. If the device running the LN software is lost or destroyed, the user loses the access to their funds. In this case however, it would be advantageous for both parties for the counterparty to close the channel, thus returning the all the funds back to the on-chain Bitcoin addresses. Also, backing up the channel mitigates this problem.

2.4 Existing implementations

In literature it is often stated that the main enabler of M2M and IoT micropayments is blockchain or other Distributed Ledger Technology (DLT) solutions [10] [1] [9]. Therefore multitude of research has been made on the subject using DLTs to facilitate M2M and IoT micropayments. Some of the papers on the subject utilize the Bitcoin LN as the payment method [1] [9], while others use some other type of DLT solution. One example of a DLT that is not a blockchain is IOTA, which based on a type of DLT called Directed Acyclic Graph (DAG). IOTA is a system that is designed for value and data exchange in IoT ecosystems. They claim that payments in their system are feeless, but the transactions are not confirmed instantly, but take some minutes to be confirmed [56]. Yet another approach is presented in a recent paper [57] a new blockchain architecture for micropayments is designed. Instead of the trustless model, this blockchain relies on trusted third parties. At the time of writing, no research could be found that addresses payments specifically in a MAS. Most of the research focuses on payments in IoT. In this section a few of the most relevant DLT based micropayment proposals for IoT or M2M are presented.

A comprehensive study [1] on finding and optimizing a blockchain based micropayment solution for IoT ecosystems concluded that LN is a good choice as a micropayment system compared to multiple other blockchain based solutions. The study was part of EU funded bIoTope project [58] that aims to promote interoperability IoT ecosystems, as currently a big problem in IoT is that often each company has their proprietary system that is not interoperable with systems from other vendors. The research paper consists of three parts: making an analysis between different micropayment solutions, designing a framework for integrating LN into existing IoT ecosystems and creating a new algorithm for LN fee reduction. In the first part they review 19 different papers considering integrating DLT based systems to IoT systems. Only few of these papers tackle the micropayment problem, but most rather concentrate on smart contract logics. From the literature review as well as an experimental analysis of a few blockchain based payment solutions the conclusion is that for purely micropayment functionality LN is a serious candidate for implementing an efficient micropayment solution for IoT systems. Therefore, the rest of the paper focuses on integration and optimization of LN into IoT setting. The integration of LN is designed to be done for a digital marketplace for the IoT ecosystem envisioned by the bIoTope project. The marketplace would allow for buying and selling data between different IoT solutions. The payments for the data would be done over LN. An algorithm is also developed that demonstrates cost savings compared to naive LN usage. The cost savings are higher when the algorithm is run longer time. The paper concludes that “off-chain” technologies such as LN are a viable solution for achieving micropayment functionalities in IoT ecosystems. The affordability and speed for the payments are common objectives

between the reviewed paper and this thesis. Therefore, the paper validates the choice of LN as the selected micropayment infrastructure for the M2M payment design proposed in this thesis.

In the paper from Kurt et al. [9], a solution is proposed to enable constrained IoT devices to make payments over the Bitcoin LN. They explain that constrained devices are often not adequate in hosting a LN node themselves. Therefore, they propose a secure and efficient solution where a LN gateway hosting a LN node is used to relay the payments initiated by the constrained device. Even without the contribution of the authors, the functionality is trivial to implement as long as the IoT device and the gateway fully trust each other, for example in a situation where they are controlled by the same entity. The solution proposed in the paper enables a situation in which the IoT device and the LN gateway are not controlled by the same entity. Instead, the provider of the LN gateway is a separate entity that provides the gateway service for a fee. This makes it easy to employ IoT devices as the entity managing them does not need to manage a full LN node on top of managing the IoT system. As described in earlier section, an LN channel is formed by creating a multisignature Bitcoin transaction between the participants of the channel. The proposal in the paper is to modify this multisignature transaction to also contain the signature of the IoT device on top of the two existing signatures. This enables a very secure and possibly even trustless setup. The disadvantage of this setup is that it makes changes to the LN base protocol and therefore requires the gateway LN node and the node that it forms the channel with, to run modified versions of the LN node software. Alternatively, the proposal could be included in the default LN implementation. The proposal is interesting and if it attains widespread usage, it could enable an easy way for IoT providers to facilitate a micropayment functionality using LN without needing to be concerned about the LN setup and management themselves. They could instead attain a relatively cheap and easy access to the LN directly from constrained devices without hosting an LN node themselves.

An IOTA based M2M micropayment system is proposed in a paper from 2018 by Strugar et al. [4]. They propose a system for automatically billing the charging of Electric Autonomous Vehicles (EAVs). The proposal consists of using IOTA Flash Channels to make real-time streaming of payments as the charging of the EAV progresses. This means that multiple very small payments are made, such that the amount of electricity charged constantly corresponds to the amount that is paid. Message Queuing Telemetry Transport (MQTT) is used to transmit the real-time payments in the Flash Channel. IOTA Flash Channels are type of payment channels for IOTA similar to LN payment channels on Bitcoin. When the charging is completed, the Flash Channel is closed, and it generates a corresponding IOTA transaction to the IOTA DAG. The authors developed a proof of concept implementation where they connected a temperature sensor simulating the charging sensor to a Raspberry Pi and were able to make IOTA Flash Channel transactions based

on the temperature indicated by the temperature sensor. The proposal uses the IOTA cryptocurrency as the payment, so the user is exposed to the price fluctuations of IOTA. If IOTA finds widespread usage, the security of the DAG system is proven and the price volatility concerns can be mitigated, then this type of payment solution could be a plausible solution for M2M micropayments.

3. DESIGN OF THE PROTOCOL

Prior to the implementation of the protocol, its functionalities must be properly defined and designed. Setting the requirements and specifications as well as in depth design of the protocol are outlined in this chapter.

3.1 Requirements and specification

To define the scope of the implementation some functional and non-functional requirements are set for the implementation. The system is not intended to be used in a production environment, but as a proof-of-concept design, so the requirements do not need to encompass everything that would be needed for a system that is intended for business use. Particularly multiple non-functional requirements related to areas such as security, scalability and device compatibility are left outside of the scope of this thesis. Instead, aspects that are fundamental to the micropayment functionality are considered. As the system is intended to be a general proof-of-concept system for several use cases of micropayments, the system should be designed such that it could be used in multitude of use cases without any changes to the core protocol and without major changes to the implementation.

One fundamental, maybe even self-evident, aspect of a payment system is the trustworthiness of the payments. This means that when the transacting parties both agree that the payment is processed, it should be definite that the payment is actually processed and is non-reversible unless agreed by both parties. Bitcoin, including LN is said to be *trustless*, meaning that neither transacting party should need to trust an intermediary nor the other party to be honest during the transaction [11][18]. This is actually even stronger guarantee than mere trustworthiness, which could refer to trusting a trusted third party. This trustlessness is added as a non-functional requirement since it should be achievable, because of the trustless nature the LN.

Some of the other non-functional requirements arise from the research questions. For the payment protocol to be suitable for M2M micropayments, the payments should be fast and cheap. These factors are added as non-functional requirements. As the implementation is built on the FIPA MAS framework, the implemented protocol should fully comply with the FIPA specifications. The scope of the thesis also requires that the value is sent over

the Bitcoin LN.

The non-functional requirements for the implementation are listed below.

1. The protocol and implementation should be easily adaptable to various micropayment use cases.
2. The time from initiating the payment to the moment when both parties agree that the payment is processed should be minimal.
3. The transaction fee should be minimal.
4. The nature of the payments is trustless.
5. The protocol and implementation are FIPA-compliant.
6. The value is transferred using Bitcoin Lightning Network.

The functional requirements arise from the use case of the implementation. One obvious functional requirement is that the participants should be able to send value to each other. Realistically in most payment scenarios the amounts in payments would be denominated in traditional currencies like euro instead of bitcoin. Because of this the designed protocol should use traditional currencies as a basis unit for the payment. To achieve this, before the transaction is made, the payment amount that is denominated in a traditional currency should be converted to an amount of bitcoin that both parties agree. To demonstrate the ability of negotiating the payment amount in a traditional currency, it is adequate to support denomination in only one traditional currency.

The value of bitcoin in traditional currencies is not constant as the value fluctuates based on the bitcoin market price set on the global free market. Because of this the participants might not agree on the exact amount of bitcoin to be transferred over the LN. This is because the parties might fetch the current price of bitcoin from different sources or at slightly different moments. To overcome this potential complication during the negotiation of the transaction, a price tolerance can be added. This means that even if the parties arrive at different amounts of bitcoin that should be transferred, they can still agree that the amounts are close enough for the payment accepted by both parties. Different use cases might require different tolerances, so the tolerance should be configurable.

To be able to evaluate the performance of the system regarding the payment time and fees, logging of these values should be included. It is meaningful to separately evaluate the time it takes for the whole payment process in contrast to the time it takes for the actual payment on the LN. This information is important to identify the possible bottleneck regarding to the payment time.

The functional requirements for the implementation are listed below.

1. A possibility to send value from the sender to receiver. The amount should be agreed to be correct by both parties.

2. The agreement of the amount of value sent is denominated in a traditional currency such as euro.
3. It is possible to define an accepted tolerance in the amount of value sent.
4. Has a functionality to display the fees and time spent for each payment.

The defined functional and non-functional requirements set a clear scope for the design of the system.

3.2 Agent analysis and design

To facilitate various interoperable implementations of the system, it must be unambiguously defined and designed. This section aims to define explicit communication patterns between the agents to facilitate definite guidance for the implementation and in theory make it viable to create multiple interoperable implementations.

JADE is chosen as the development framework to implement the system. It is the most established of the FIPA-compliant agent development environments. As JADE is developed in the Java language, the implementation is also written in Java. The best practices and constructs of Java and JADE are considered when designing the system.

During the design phase, a methodology proposed in [59] is loosely followed. The methodology is intended to be used for the design of a full scale MAS with a specific use case, in contrast to the system devised in this thesis. Therefore, only applicable aspects of the methodology are taken into consideration. The methodology is divided in two main phases: analysis and design. The analysis part covers defining the use cases, agent types, agent responsibilities and agent relations. Information about agent deployment is also a part of the analysis phase, but it is not relevant for the scope of this thesis. The design part covers specifying the interaction protocols, message templates, agent-resource interactions, internal behaviours, defining an ontology and content language selection. Agent discovery mechanisms would also be included in the design phase, but that is not relevant for this thesis.

The use cases of the system are relatively simple. The main use case is to send payments between agents. Therefore, there should be an agent that can send payments and an agent that can receive payments sent by the first agent. This simple description also forms a base for agent types, agent responsibilities and agent relations. The identified agent types are payment sender agent and payment receiver agent. Their responsibilities are self-evident: the payment sender agent makes a payment and the payment receiver agent receives the aforementioned payment. The relation of the agents is also clear: they interact between each other. The use cases could be made using the preliminary information, but it is not relevant in this case, since there are no external users to the system and the use case is well defined.

To satisfy the requirements set in Section 3.1, the agents must interact with some resources external to the MAS. To satisfy the requirement that the payment must happen via the LN, the agents need to interact with the LN. The sender agent must interact with it to send a payment and the receiver agent must interact with the LN to convey the information for the payment to the sender. The only way of achieving a full access to the LN is through an LN node. Therefore, the LN resource is labelled as the LN node.

Another requirement that necessitates external resources is that the value of payment should be denominated in a traditional currency. As the monetary unit of LN is bitcoin [48, Chapter 3.], it is not possible to derive the price of a payment in traditional currency by interacting only with the LN. Because of this, an external bitcoin price resource is used. From now on, this resource is also labelled the *Price Application Programming Interface (API)*.

Based on earlier analysis, an *agent diagram* is presented in Figure 3.1. The agent diagram follows the format presented in [59]. The circles represent agents and rectangles represent resources that are external to the MAS. Arrows mark interactions between elements. Even though only one of each resource is portrayed, the agents might interact with different instances of the resource. For example, the agents might use different APIs acting as the bitcoin price resource. The LN nodes that the agents interact with are also separate.

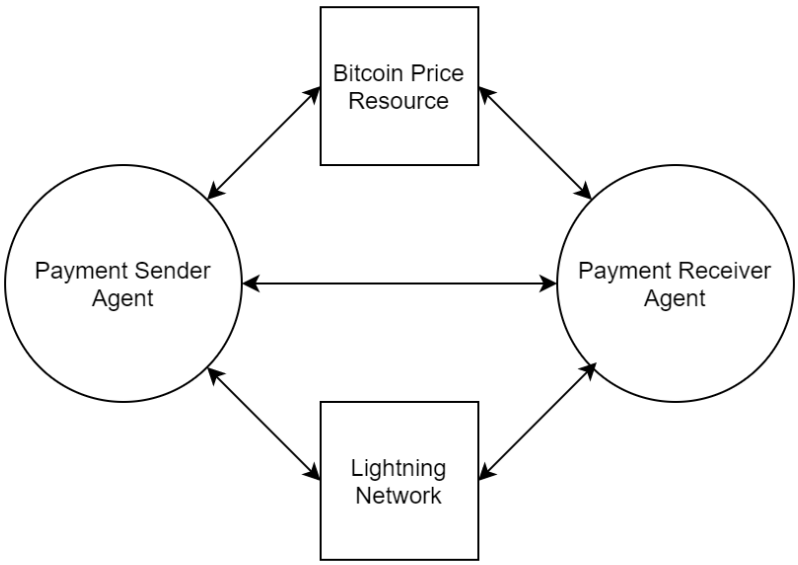

Figure 3.1. *Agent diagram of the system.*

In the design phase an important step is to define the interactions between the agents [59]. In FIPA-compliant systems, agent interaction is often defined using Agent Interaction Protocol (AIP). An AIP describes the communication pattern and the semantics of the

communication for a specific type of communication. The protocol must use and comply with the CAs defined in FIPA ACL specifications. The FIPA specification includes multiple predefined AIPs that cover many general types of interactions. [60] [61] [62]

The methodology [59] states that standard FIPA AIPs should be used for the interaction, unless no suitable AIPs exist. The predefined AIPs do not include a protocol for monetary transactions, so a new ad-hoc AIP for monetary payments is proposed as suggested by the methodology. The ad-hoc AIP devised in this chapter is designed specifically to employ LN payments between agents. The AIPs are typically represented using the Agent UML (AUML) sequence diagram [63]. AUML is a modelling notation for modelling MASs. AUML extends the notation of UML 2.0 to be compatible for agent to agent interactions. The AUML sequence diagram, which is an extension of the UML sequence diagram, defines the participants of an interaction, the messages sent during the interaction and the order and participants of said messages [63] [36]. To specify the AIP for the LN payments, the AUML sequence diagram is used. The AUML sequence diagram for the protocol is presented in Figure 3.2. The interactions that are external to the agents are not presented in the AUML diagram, since an AUML sequence diagram only concerns the interactions between agents. The diagram is extended in a sequence diagram in Figure 3.3 to include all the interactions between different elements of the system. The diagram in Figure 3.3 is a sequence diagram adopting components from AUML and UML sequence diagrams. In the diagram, the sender of the payment and the receiver of the payment are isolated in their respective containers to signify the components belonging to each one. Each counterparty has their individual instances of the bitcoin price API and the LN node. The diagrams are further explained in the following paragraphs.

The AIP consists of a sequence of messages between interacting agents [60]. These messages are CAs defined in [61]. The requirements that were defined in Section 3.1 pose guidelines and constraints for designing the AIP. Particularly to satisfy the requirements of *trustlessness*, *denomination of value in a traditional currency* and *adaptability to various micropayment use cases* must be considered. The requirement of trustlessness means that the counterparties do not need to trust each other to act honestly. This requirement is satisfied, if the receiver can independently verify that they receive the payment. The requirement of denomination in a traditional currency imposes a need for the agents to convert the monetary value of the payment to corresponding value in bitcoin. To do the price conversion in a trustless manner, the agents need to verify the price independently. The adaptability to various use cases can be facilitated in several ways. Usually a payment is a payment for a product or service. For the receiver of the payment, which is usually a seller of the product or a service, it is meaningful to associate the payment to the product or a service that they are selling. Another aspect that could facilitate multiple use cases is an ability for the receiver to optionally decline a payment based on conditions that are specific for that use case. To facilitate these features, a textual identification for

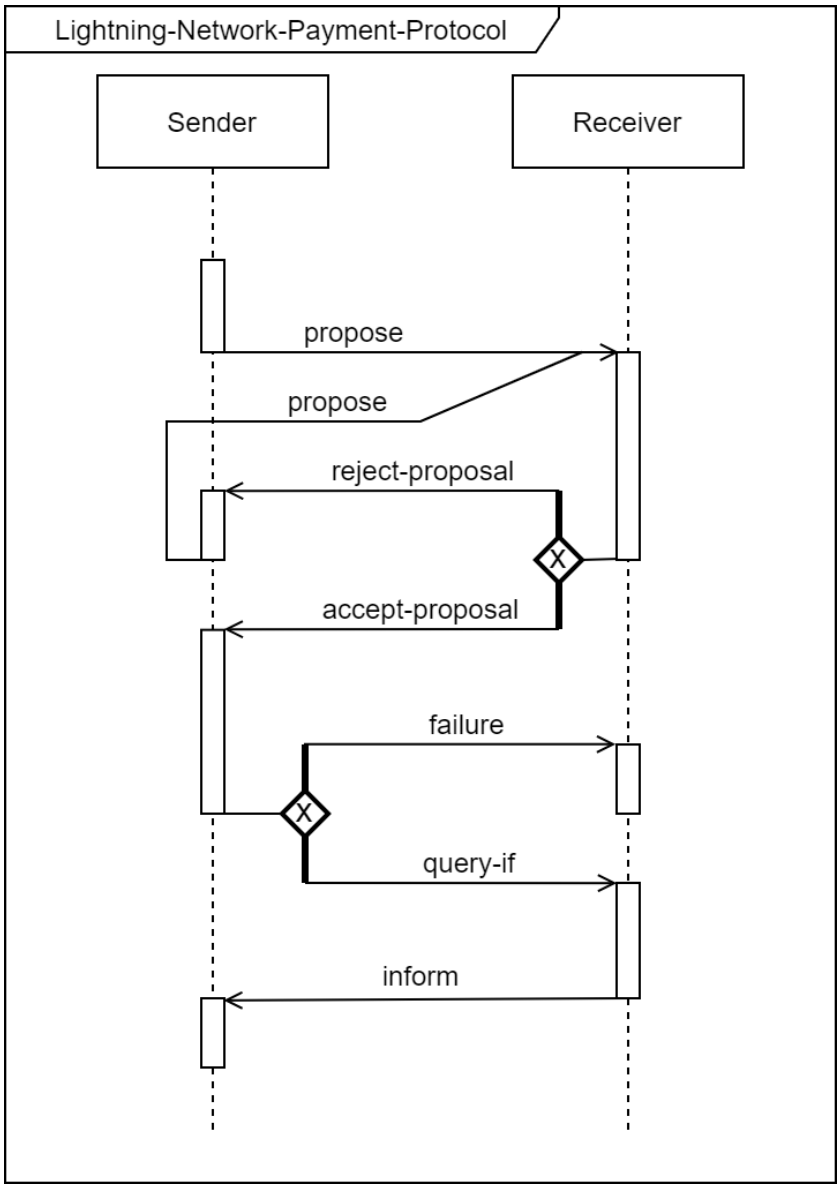

Figure 3.2. *AUML sequence diagram of the payment interaction.*

the payment could be integrated to the AIP. The identification is labelled as the *payment identifier*.

In FIPA AIPs one agent is always the *initiator* who sends the first message, and thus initiates the AIP [64]. In the case of a payment protocol, it is not evident if it should be the sender or the receiver of the payment who should initiate the interaction. In some use cases it would be meaningful for the receiver to request for a payment from the sender. This is analogous to sending a traditional invoice to the entity who is supposed to make the payment. In some cases, the payment initiates the whole process for buying a product or service. In this case it is more suited that the sender of the payment is the initiator. The sender is chosen as the initiator for the AIP. Therefore, both terms, *initiator* and *sender* used in this chapter refer to the same agent. In the use cases where it would be more practical for the receiver to start the interaction by requesting a payment, the request can

be done in advance, before initiating the protocol. The sender of the payment can then seamlessly respond to the request by initiating the protocol.

Some preconditions are specified that must be met before the AIP can be initiated. To start the interaction there must be a preliminary agreement that a payment can or should be made from the sender to the receiver. This agreement depends on the use case. In some cases the agreement is such that the receiver accepts any payments, but in other cases the sender may need an explicit permission from the receiver to be allowed to initiate the interaction. The payment identifier mentioned earlier, must be known by the initiator before initiating the interaction. By using the payment identifier, the receiver can restrict the incoming payments to only the allowed payments. To facilitate this restriction capability, the payment identifier must be included in the initiation message. The usage and format of the payment identifier is profoundly dependent on the use case. It can be used to restrict payments, to associate payments with products, to authorize counterparties or to facilitate the accounting of the payments. Depending on the use case, the payment identifier could be publicly announced by the receiver, generated by the sender or generated by the receiver and conveyed to the sender before the initiation of the protocol.

Besides the payment identifier, some other values must be defined in the initiator agent before the initiation. To satisfy the requirement of minimal duration of the interaction, the amount of messages exchanged during the interaction should be minimized. Therefore the initiator should include in the initiation message all the information that the receiver might need in verifying the payment. This includes the amount of value to be sent denominated in a traditional currency as well as in Bitcoins. This requires the initiator to fetch the current price of bitcoin already before sending the initiation message.

Making payments in the LN is usually not free, since often nodes that the payment passes through charge a fee to route the payment [51]. The fee is charged in a way that the amount that the sender of the payment pays is larger than the amount that the receiver receives. The difference is paid as a fee to the routing nodes. Therefore, it is meaningful to set a maximum fee that the sender is willing to pay for the payment. Before initiating the interaction, this fee should be defined for the initiator agent.

To preserve the trustless element of the interaction, both parties independently fetch the price of bitcoin from their trusted price APIs. Depending on the price resources, the prices fetched may not be identical. To allow for this deviation in prices, a price tolerance must be defined for the receiver of the payment. The tolerance is the maximum allowed deviation of the proposed price relative to the price fetched by the receiver.

When all the mentioned values and parameters are set, the sender can initiate the protocol by sending the first message. Each FIPA message corresponds to one of the CAs [61]. The CA chosen for the first message is *Propose* as the initiator proposes to make

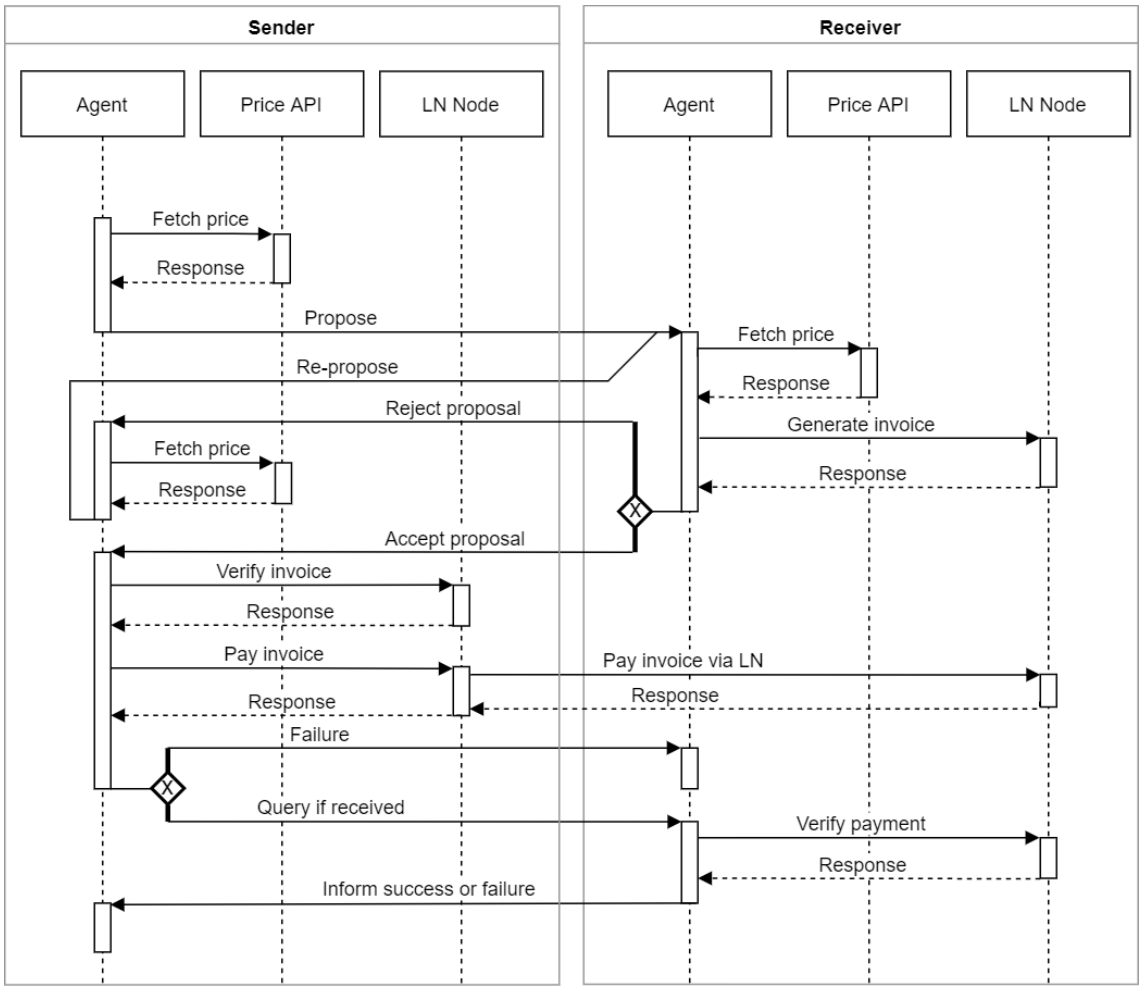

Figure 3.3. *A complete sequence diagram of the interaction including the interactions with external resources.*

a payment. The message contains the payment identifier, the value of payment in bitcoin and the value of payment in traditional currency. The unit of traditional currency must also be included. ISO 4217 currency codes [65] are used as the unit. These include for example *EUR* for euro and *USD* for United States dollars. The payloads of each message in the protocol is presented in Table 3.1.

After receiving the proposition for the payment, the receiver first verifies that it accepts the proposition. The verification consists of steps that are partly use case specific. The checks that are not use case specific, but always verified consist of two separate verifications. First is verifying that the general structure of the initiation message corresponds to the protocol. The second is to check that the bitcoin price proposed by the initiator is correct. For this the agent independently fetches the bitcoin price from the price API and compares this price to the price proposed by the initiator. If the prices correspond in limits of the specified tolerance, the price-check passes. The use case specific verification can be specified separately for each use case. This facilitates the usage of the implementation for multiple use cases. The use case specific verification is mainly envisioned to

make use of the payment identifier. The agent can choose to accept incoming payments with only certain payment identifiers. Also the amount that the initiator proposes to pay can be used to determine if the proposition is accepted. The conditions for the use case specific verification should be defined before the protocol is executed.

If all the verifications pass and therefore the receiver accepts the proposition, they reply the initiator confirming that the payment proposal is accepted and that the initiator is allowed to make the payment. The CA of the reply is *Accept Proposal* as that is the standard affirmative response to a propose message in a FIPA-compliant protocol [61]. To make the payment, the payer must know the information required to make the payment, such as the address of the receiver. In LN the receiver usually conveys the aforementioned information to the payer as an LN *payment request*, often called an *invoice* [53]. The invoice is an encoded string of characters that contains all the required information to make the payment. The invoice is included in the reply message, so that the payer can immediately make the payment over the LN without separately requesting the invoice.

The invoice contains a field to include a description for the payment. This description can be used to connect the LN payment to the AIP. With it, it is possible to verify which payment in LN corresponds to the instance of AIP. This enhances the trustless aspect of the protocol. An instance of a FIPA AIP has a unique identifier called *Conversation Identifier*, that persist throughout the AIP [66]. The payment identifier as well as the conversation identifier are included in the LN invoice description. The amount in traditional currency and the unit of the currency are also included in the description. As the price of bitcoin varies a lot, it is cumbersome to deduce the value of a historical bitcoin payment in a traditional currency. By including the value of the payment in traditional currency, it is directly visible by browsing the payments on the LN.

If the receiver does not accept the proposal or an error arises in the process of creating the invoice, the receiver responds with a rejection message. This rejection represents the first of several possible error situations in the protocol. The CA of the message is *Reject Proposal*. In the rejection message, the receiver can include a reason for the rejection. The sender can then try to reinitiate the protocol with a propose message identical to the original initiation message. The sender can make modifications to the initiation message based on the reason of rejection. For example, if the reason for rejection is that the bitcoin price tolerance was surpassed, then simply fetching the price again might solve the issue. There should be a limit imposed on the amount of reinitiation attempts to prevent the formation of an infinite loop.

Once the sender receives the accept proposal message containing the invoice, they check if the invoice is correct. The encoded invoice can be decoded such that it is possible to examine the components of the invoice [53]. Before making the payment based on the invoice, the sender should check that the amount corresponds to the original proposition.

Table 3.1. *The payloads of messages of the Agent Interaction Protocol*

Communicative act	Sender of the message	Payload
Propose	Payment sender	- Payment identifier - Amount in bitcoin - Amount in traditional currency - Unit of traditional currency
Accept Proposal	Payment receiver	- Indication of acceptance - Lightning Network invoice
Reject Proposal	Payment receiver	- Indication of rejection - Reason for the rejection
Failure	Payment sender	- Indication of a failure
Query if	Payment sender	- Lighting Network payment hash
Inform	Payment receiver	- Indication that the payment is received

The sender also verifies that the description of the invoice corresponds to the protocol and contains the correct conversation identifier, payment identifier and amount in traditional currency. If the verification of the invoice is successful, the sender makes the payment over the LN. LN functions in a way that once the sender of the payment gets a confirmation of a successful payment from their LN node, the payment is settled and the receiver can instantly verify that they have received it [48, Chapter 3.]. Once the payment has completed, the sender knows that the receiver has received it.

If verifying the invoice fails or making the payment over the LN fails, the sender informs the receiver that there was a failure. This is the second possibility an error during the interaction. A failure at this point concludes the protocol. It would be possible to retry the initiation, since at this point both parties still agree that the payment is not made. However is not meaningful to retry at this point, since the error probably originates from the LN. The error could for example be an error finding a valid payment route [48, Chapter 3.], which cannot be fixed without adding complex LN node management functionalities for the agents. This kind of functionality is outside the scope of this thesis.

To conclude the interaction and form an agreement between the agents in which both know that the counterparty agrees that the payment is made, they exchange final messages. Instantly after the sender has made the payment, they demand the receiver to verify that they agree that the payment is settled. This is done via a message with a CA of *Query If*. The message contains a unique identifier of the payment called the *payment hash*. The receiver can use this to identify the payment on the LN.

As the receiver receives the Query If message, they check that they have received the payment using the payment hash provided in the message. This step is important, since without it, the sender could claim that they have sent the payment, even though they

have not. If the receiver agrees that they have received the payment, they respond with a message with a CA of *Inform* to confirm that they have received the payment. Once the sender of the payment receives this confirmation message, both agents are in agreement that the payment is concluded. At this point the protocol ends and the agents can proceed to perform use case dependent actions that are executed after a successful payment. They could for example record the payment to a database or proceed to a following interaction.

If the receiver cannot verify that they have received the payment a third type of error situation arises. This is the most critical error, since at this point the sender and the receiver disagree on the state of the payment: the sender claims to have sent the payment, but the receiver believes not to have received the payment. As the LN functions in a way that as the sender gets a confirmation of a delivered payment, it is not possible for the receiver not to have received it [48, Chapter 3.]. Therefore, there are only few circumstances on how this kind of situation can arise. One possible reason for this is that either the sender of the payment or the receiver is a malicious actor and is not being honest about the status of the payment. Another possibility is that the receiver loses the connection to their LN resource, thus having received the payment in reality. These kinds of problems cannot be managed by the protocol. Both agents should rise a serious error at this point, so that the owners of the agents can handle the dispute.

The methodology [59] states that defining message templates, agent-resource interactions, internal behaviours and an ontology as well as content language selection should also be done in the design phase. The agent-resource interactions were already preliminarily described during the description of the AIP, and will be further addressed in Chapter 4. Some of the internal behaviours that are relevant for the protocol, such as validations, were also addressed on high level. They are elaborated in Chapter 4. Message templates, defining an ontology and content language selection are also addressed in Chapter 4.

3.3 Interfaces

There are two separate external resources that each agent must interact with. These are the bitcoin price resource and the LN resource. The bitcoin price resource is an API from where the current bitcoin market price can be retrieved. The LN resource is a connection to an LN node controlled by the same entity that controls the agent. As the agents might be located on constrained devices, it must be possible to locate the resources on separate physical devices. The methodology [59] that was used in analysing and designing agents suggests an approach for managing the external resources where each resource corresponds to a separate agent, whose only purpose is to interact with the resource. This approach would unnecessarily complicate the system. Therefore the

system is designed in a way that the resources are directly interacted with from the sender and receiver agents.

The bitcoin price resource is used to retrieve the current price of bitcoin. To ensure the correct functionality of the overall system the interaction with the price resource should be relatively quick, it should be possible to retrieve the price in different currencies and the price that is fetched should correspond well to the overall market price. In contrast to most stocks, the trading of bitcoin is fragmented to multiple exchanges around the world [67]. Thus, there is no absolute official price of bitcoin that can be referenced. Each exchange has their distinct rate, which might differ from other exchanges. Nevertheless, a concept called arbitrage makes sure that the prices across exchanges stay relatively stable [67]. Arbitrage is an act of simultaneously buying and selling the same asset on different exchanges, such that definite profit is produced for the actor. Besides generating profit, arbitrage stabilizes the prices across exchanges. For bitcoin the price stays generally relatively stable, except at extreme market conditions events where the price moves quickly. For example, during the year 2019 the price differences between exchanges remained mostly within 0.5 % of each other [67]. Because of the price stability, the price can be obtained from any public API that is deemed reliable. For example, some exchanges as well as some other webpages offer public free APIs supplying the current bitcoin price. Most of these sources provide the price denominated in multiple currencies. For additional reliability, multiple sources could be used simultaneously and an average or median price of those could be used. To satisfy the non-functional requirement of adaptability for different use cases, it should be made possible to easily adopt different APIs.

The second external resource is the LN node that is used to interact with the LN. LN node is a software that is used to send and receive payments in the LN [48, Chapter 4.]. A node is also used to hold funds either in LN payment channels or in an on-chain Bitcoin address. To be able to interact with the node from an agent, the node must provide an API that can be connected to from the agent. As previously stated, the node should be able to exist on a separate device, so the API must support connections across devices. As the connection to the node might even be conducted over the public internet, the connection should be strongly encrypted and authenticated.

If a single entity, such as a corporation, controls multiple agents, it is not required that there is a separate LN node for each agent. As long as sufficient monetary and compute capacity is maintained on the node, a single node could possibly manage even multiple concurrent payments from different agents. Figure 3.4 illustrates a setup where two companies are involved in a payment interaction. Each company controls multiple agents that are connected to the single node owned by the company. Each stick figure represents a separate agent. Solid arrows represent agent to LN node connections that happen within the company and the dashed arrows represent the communications between the two companies. One of the dashed lines is the agent to agent communication and the

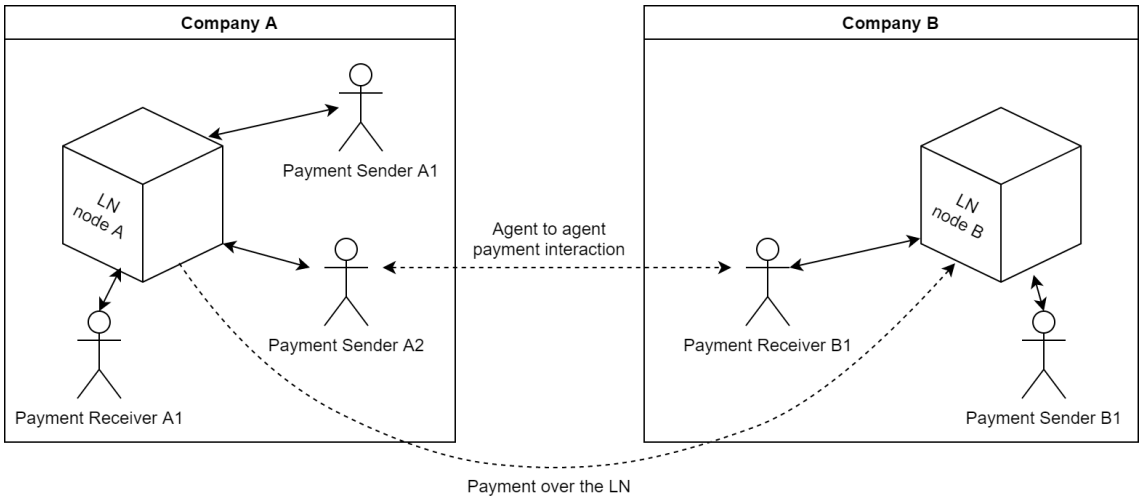

Figure 3.4. *An example of agent and LN node connections between two companies.*

second one is the payment over the LN. Even though both communications are part of the same payment interaction, they are displayed as separate communications as they are transmitted via separate protocols. The agents that are not shown to communicate with the other company could be in fact communicating with a third company. The amount of agents could also be changing during the execution, such that new agents are created or old ones destroyed.

4. IMPLEMENTATION

This chapter outlines the implementation of the system that was designed in Chapter 3. The chapter is divided in four parts. First is the setup and configuration of the LN nodes that are used to make the payments on the LN. Second is the implementation of the agents and the protocol using JADE. The third section focuses on the interfaces that are used by the agents to interact with the LN nodes and the Bitcoin Price API. In the last section, the implementation is tested in an example use case, by complementing pre-existing JADE agents.

4.1 Lightning Network setup

An access to a LN node is needed to be able to send and receive funds on the LN. There are three major LN node implementations by three different companies [48, Chapter 4.]. The code for all of the implementations is open sourced. The implementations are c-lightning by Blockstream, Lightning Network Daemon (LND) by Lightning Labs and Eclair by ACINQ. Any of the three LN nodes could be used, as each of them exposes an API through which the node can be operated. LND is chosen as the LN node to be used, since it is the most popular of the three and offers extensive documentation [68][69]. LND exposes a gRPC Remote Procedure Call (gRPC) API, which is a modern Remote Procedure Call (RPC) framework [70]. As mentioned in Section 3.3, on many of the possible use cases, the LN node might be located on a device that is distinct from the device where the agent is located. To simulate this situation, the LN node is installed on a *Raspberry Pi 3 Model B+* device that is located on the same Local area network (LAN) as the computer where the agents are developed and tested.

The LND node, such as all the other LN nodes, require a connection to a Bitcoin node [71][68]. The connection is needed to access to obtain a direct and secure access to the Bitcoin blockchain. A Bitcoin node requires over 400 GB of hard drive storage and requires relatively heavy computing capacity, so a Bitcoin node is not installed on the Raspberry Pi but instead on the same computer as the agents. The Bitcoin Core version 0.21.1 [72] is used as the Bitcoin node. A program called bitcoind is a component of the Bitcoin Core, that exposes an API that can be used to connect to the node [73]. After installing the Bitcoin Core, the software automatically starts downloading and validating

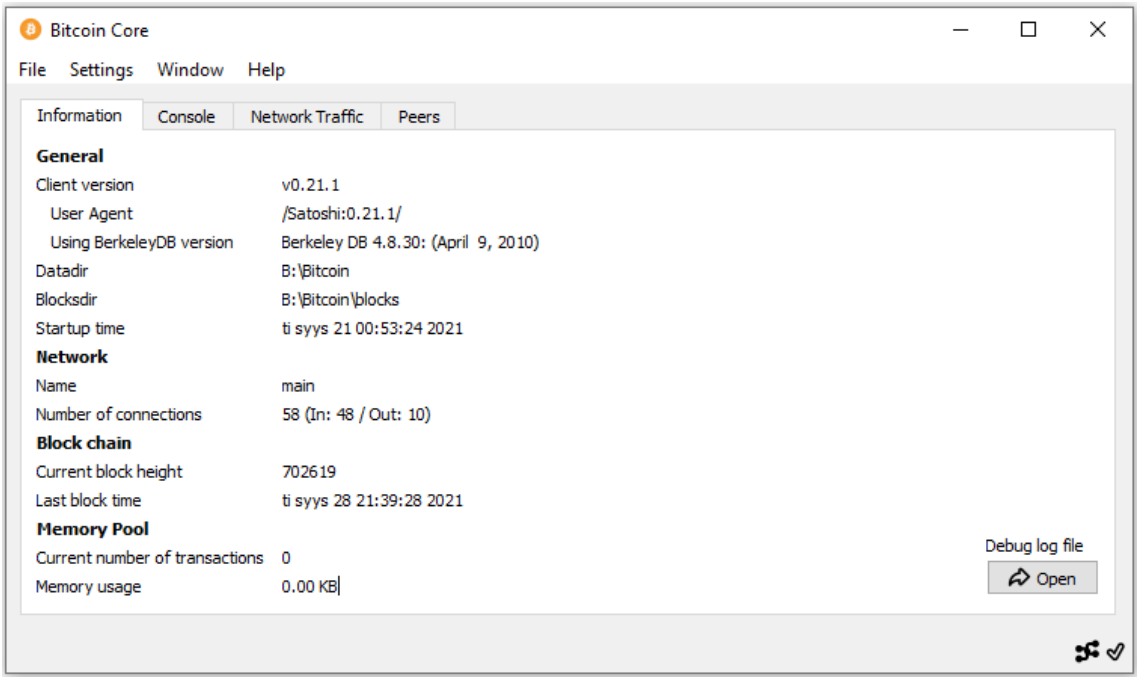

Figure 4.1. *A screenshot of the Bitcoin Core program.*

the Bitcoin blockchain, containing the complete transaction history of Bitcoin. The Bitcoin node is fully operational only after the blockchain is fully downloaded and validated. Figure 4.1 presents a screenshot of the Bitcoin Core program in Windows environment. The checkmark on the bottom left corner of the window implies that the blockchain is fully downloaded and validated.

To allow for the LN node to connect to the Bitcoin node, some configuration needs to be made. Bitcoin Core contains a configuration file, *bitcoin.conf*, that is used to configure the program. Notably configuration entries are made that allow API connections from external devices. Bitcoin Core uses ZeroMQ Transmission Control Protocol (TCP) communication library to broadcast events happening on the blockchain [74]. ZeroMQ broadcasting uses publish-subscribe pattern to notify the subscribers of the events happening on the blockchain. The LN node subscribes to the ZeroMQ messages to remain up to date on the state of the blockchain [71]. The publish-subscribe pattern is used, so that the LN node does not need to keep polling the Bitcoin node. Entries are made in the Bitcoin Core configuration file to enable the ZeroMQ broadcasting to external devices on the same LAN. To support the Bitcoin network, a network port to the Bitcoin Core is opened on the router that connects the LAN to the internet [75].

For the installation of the LN node, three different guides [71][76][77] are followed and applied. Two separate LN nodes are needed to have a separate node for the receiver and the sender of the payment. LND facilitates running multiple isolated instances on the same device simultaneously. LND operates is such a way that as the LND instance is launched, a configuration file can be provided as a command-line argument. To run

multiple separate instances of LND, a separate configuration file is provided for each instance. Two instances, with two separate configuration files are launched.

Most of the entries in both configuration files are identical. The basis for the configuration files is adopted from an LND setup guide [76] written by Alex Bosworth, the Lightning Infrastructure Lead at Lightning Labs. Some additions are made to the reference configuration. For the external program, in this case the agent, that is using the LND node via an API, the authenticity of the LND node is confirmed using a certificate file. Each LND instance could provide a separate certificate that could be defined in the configuration file, but in this case a common certificate is used. The certificate is automatically generated to the LND default data directory. The Internet Protocol address (IP address) of the device on the LAN is added to the configuration file to be included in the certificate file. The connection to the Bitcoin Core node is configured by adding the IP address, username and password of the Bitcoin Core API. The addresses are added to where Bitcoin Core broadcasts the ZeroMQ messages, so that LND can subscribe to the messages. The same Bitcoin Core node can be used for both LND instances as it is only used to fetch and relay information to the blockchain. In this configuration the Bitcoin Core node does not keep any information about the user, their addresses or transactions. All of the persistent data is contained in the LN nodes.

Some entries, that are distinct in each LND node, are added to the individual configuration files. A separate data directory has to be assigned for each LND instance, thus distinct data directories are defined in each configuration file. Another essential configuration that must be distinct for each instance are the network ports. There are two incoming network connections for each LND instance. These are the RPC connection for the agents and the LN network connection that is used to communicate with the LN peers. These ports are specified in the configuration file. The connections incoming from the LN peers to the specified ports, come from the public internet. Therefore, the ports from the router, that is connecting the LAN to the internet, must be opened and directed to the corresponding ports on the Raspberry Pi. After all the configuration is finished, the node instances are launched by providing separate configuration files as command-line arguments. The configuration entries that are not specified in the distinct configuration files are read from the default common configuration file.

The LND nodes are operated by using *lncli*, which is a command-line tool that can be used to send commands to the LND node [78]. All the interactions with the LND node in this section are performed using lncli commands. lncli uses the gRPC API to connect to the node. When executing commands on the lncli, two command-line parameters must be provided. The gRPC port must be provided for the lncli to be able to connect to the correct node instance. Secondly an authorization file must be provided that authorizes the usage of the LND node. The authorization is done using a macaroon file that is generated in the data directory of an LND instance [79]. The path to this file is provided as the second

command-line argument. When using the gRPC API from a separate device, these same parameters must be provided as well as the certificate file.

Before making payments on the LN, a payment channel must be established. The easiest way to make payments between two nodes would be to make a direct payment channel between the two nodes. However, this does not correspond to a realistic situation, since the payments on the LN are mostly done via a route that consists of multiple routing nodes. In very few cases is it realistic for a node to have a direct payment channel with all the counterparties that it is making payments with. To create a more realistic situation, payment channels are established to two large public nodes. This way a payment route exists between the nodes, containing multiple intermediary routing nodes. This also guarantees that the payment travels through the internet rather that in a private LAN. With this configuration, a somewhat realistic circumstances are created in terms of the speed, reliability and fees of the payments.

A web tool called 1ML [80] contains statistics of all the public LN nodes. 1ML is used to choose the intermediary nodes to which payment channels are established from the nodes running on the Raspberry Pi. A payment channel can be opened with a single command to any public node that allows it. To open a channel, funds must be available in a Bitcoin address that is connected to the LND node. LND automatically creates a Bitcoin address that can be used to add funds to the node. Some bitcoin is sent to both of the LND nodes. LND sets the default minimum channel size to be 20 000 satoshis, which corresponds to around 10 euros at the time of writing. This amount is enough for testing micropayments, so channels of said capacity are opened. After trying to open channels with some of the largest nodes with most connections, it becomes apparent that most of these nodes have set a minimum allowed channel size to be a lot higher than the default of 20 000 satoshis. Some very large ones have set the minimum capacity as high as 0.01 bitcoin, equivalent of around 500 euros. Nodes called CoinGate [81] and southxchange.com [82] allowed opening channels of intended capacity. The nodes are operated by companies providing bitcoin payment and cryptocurrency exchange services [83][84]. Payment channels of 20 000 satoshis are opened; a channel to CoinGate from the first node and a channel to southxchange.com from the second node.

The limit on how much can be transacted on a LN payment channel is defined by the capacity of the channel. The total capacity of each of the channels that are opened is 20 000 satoshis. The capacity consists of inbound and outbound capacity. The outbound capacity provides the limit on how much can be sent using the channel, while the inbound capacity defines how much can be received on the channel. As all of the funds of the channels belong the nodes opening the channels, all of the capacity on both nodes is outbound capacity. This means that at this point, both nodes can send payments, but cannot receive any payments. To test making payments between the agents, at least one of the agents must be able receive payments. Therefore, inbound capacity must be

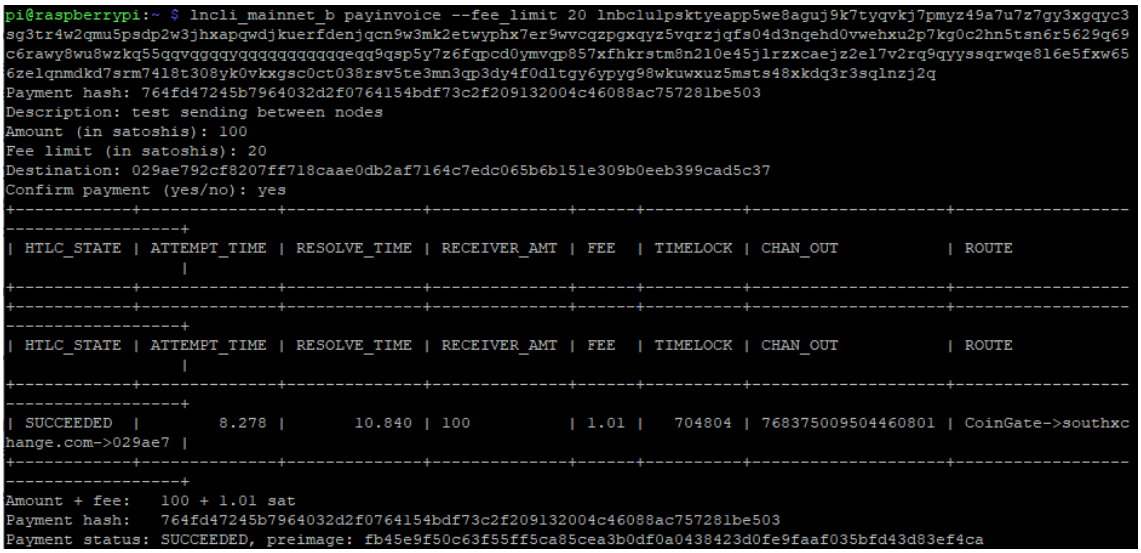

Figure 4.2. *The command and output of the payment between the two LND nodes that were setup.*

obtained on either of the channels.

There are several ways of obtaining inbound capacity on an LN payment channel. *A Builder's Guide* by Lightning Labs [85] lists three ways of obtaining inbound capacity: spending funds, using a *Lightning Loop* service and buying a channel on *Lightning Pool*. Both of the latter options are paid, but affordable services provided by Lightning Labs. The first option, spending, is used to obtain inbound capacity. An account is created to an external, free LN wallet called BlueWallet [86] where the LN node and payment channels are managed by the developers of the wallet. The developers need to be trusted to retain the funds held in the wallet, but in turn they provide an easy way of using the LN. Most importantly they manage their nodes in a way that there is inbound capacity available, thus enabling to receive LN payments to the wallet. By making a payment to this wallet, inbound capacity is obtained on a node running on the Raspberry Pi, whilst maintaining an access to the funds that are being "spent". A payment of 4000 satoshis over the LN to the BlueWallet succeeds on the first try, proving that the LND nodes are setup correctly. To test if a route is found between the nodes on the Raspberry Pi, a test payment of 100 satoshis is made between the nodes. The payment is made to the node that now has some inbound capacity. A screenshot of making the payment using the lncli tool is presented in the Figure 4.2. The command *lncli_mainnet_b* is a user defined alias for the lncli command including the required parameters: the port of the RPC API and the path to the macaroon file. A fee limit is added as an additional parameter to make sure that the fees for the payment do not exceed 20 satoshis. The long string starting with character "ln" is an invoice that was earlier generated by the receiving node. As seen from the figure, the payment route is directly from CoinGate to southxchange.com, so they seem to have a payment channel between them. This payment too succeeds on the first try.

The times it took to process the two payments were approximately 42 and 11 seconds respectively. The fee for each payment was around 1 satoshi.

The two nodes that were setup in this section are used as the nodes between which the agents make the payments. The payment that is made between these two nodes travels through the public LN as would a payment that is made between two completely separate entities located on different locations. Even though the nodes are setup on the same device, they are isolated in a way that fully simulates a realistic situation of two nodes that are owned by two separate entities.

4.2 Agent implementation

The communication pattern between the agents was defined in Chapter 3. This sets a clear outline for the implementation. However, there are multiple aspects that are not yet addressed, that must be carefully considered while implementing the system. Notably achieving the non-functional requirement of easy adaptation for multiple use cases is greatly affected by the choices made during the implementation. Also, the contents of the messages between agents must be structured in a way that the implementation is FIPA-compliant.

The implementation is done using JADE that is based on Java programming language. To ensure the quality of the implementation regarding object-oriented programming, the *SOLID* object-oriented design principles [87] are followed where applicable. SOLID principles consist of five principles of object-oriented design that are designed to establish quality of software in terms of structure and maintenance of the code. Some of these principles, such as *The Liskov substitution principle*, *The Open–closed principle* and *The Dependency inversion principle* can be used to allow modification of some elements of the implementation for the needs of different use cases.

In JADE, the agents are developed using object-oriented approach where an agent type corresponds to a Java class and an individual agent corresponds to an object of the class. Any agent that is developed must extend the *Agent* base class [88, Chapter 4.]. To be able to adopt the system for multiple use cases, *The Open–closed principle*, which is one of the SOLID principles, is used. The agents are developed in such a way that a payment receiver agent and a payment sender agent are classes that are extended from the base Agent class, and are intended to be extended in use case specific child classes. This corresponds to The Open–closed principle, since the functionality of the agents can be modified without modifying the classes that define and execute the protocol. This also hides the complexity of the whole protocol from a developer who would implement a use case specific agent.

Classes *PaymentSenderAgent* and *PaymentReceiverAgent* that extend from the Agent

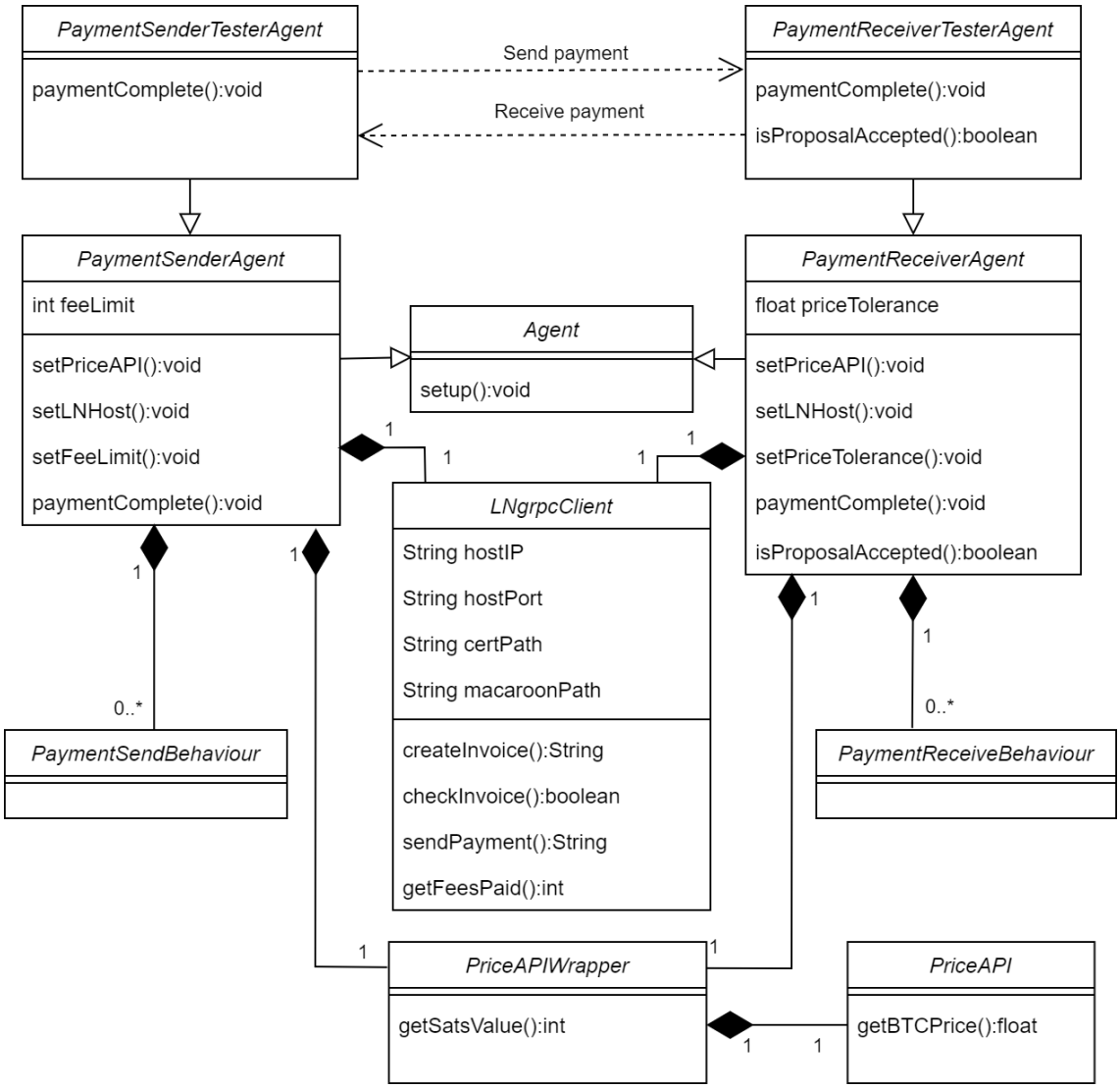

Figure 4.3. *A UML class diagram outlining the structure of the implementation.*

class, are created. From now on these classes are referenced as the base classes as they implement the payment protocol and act as base for the use case specific classes. To use the classes as intended, *PaymentSenderTesterAgent* and *PaymentReceiverTesterAgent* classes that in turn extend from the base classes, are created. The tester classes mimic the use case specific classes and are used for testing the protocol. The relation between the classes is presented in a UML class diagram in Figure 4.3. The class diagram contains the relations of all the most essential classes of the implementation. Also the most relevant methods and properties of the classes are included. To simplify the diagram, some non-essential classes, properties and methods are omitted. The diagram is further elaborated later in this chapter.

As stated, the complexity of the protocol is hidden in the base classes. In JADE the actions performed by agents are defined in behaviours. As sending a payment and receiving a payment are single activities, only one behaviour is needed for each agent: a behaviour

for sending a payment for the sender agent and a behaviour for receiving for the receiver agent. The behaviours are implemented in the base classes, and executed from the extended, use case specific classes. As can be seen from the Figure 4.3, the behaviours are classes that are contained within the agents. The execution of a behaviour is done by constructing a new instance of the behaviour class and adding them to the pool of executable behaviours of the agent. In theory multiple *PaymentSendBehaviours* or *PaymentReceiveBehaviours* can be in execution simultaneously in a single agent. However, this is not tested and falls outside of the scope of this thesis.

To make a payment, the receiver must first have the receiving behaviour running. When started, the receiving behaviour stays idle waiting for a payment proposal. The sender agent can then execute the payment sending behaviour to start the payment interaction. The agent id of the receiving agent and the information for the payment are given as parameters for the sending behaviour. The information for the payment includes the amount to be sent in a traditional currency, the currency code of the currency and the payment identifier. The sending and receiving behaviours then engage in an interaction to fulfill the payment protocol as defined in Chapter 3. Once the protocol concludes as the payment either succeeds or fails, the behaviours end, and the agents are informed of the result of the payment.

In Chapter 3 a methodology [59] for designing MAS was followed. The steps of the methodology that were not addressed in the earlier chapter are addressed in this chapter. Defining message templates, internal behaviours and an ontology, and selecting the content language are addressed in the following paragraphs.

Message templates in JADE are a tool for a behaviour to select the messages it handles. An agent may have multiple behaviours running in parallel. These behaviours could be interacting with different agents and using different protocols. Therefore, it is important that each behaviour only reacts to the messages that are intended for that specific behaviour. If a wrong behaviour reacts to a message that is not intended for it, the message is removed from the incoming message queue of the agent, and therefore is not received by the behaviour it is intended for.

The use case specific agents that use the payment protocol, might have multiple other behaviours and interactions ongoing while engaged in the payment protocol. Therefore, the message templates for the payment protocol must be well defined such that the behaviours do not process messages that are intended for other protocols. Also, a situation where multiple separate payments are made in parallel, must be taken into consideration. Each message must be processed by the instance of the behaviour that is managing the specific payment. These two factors are addressed by adding the protocol parameter of the message as well as the conversation identifier to the message templates.

To further refine the message templates, a reply-to parameter is used. This ensures that

Table 4.1. *The message templates for the messages in the protocol.*

Receiver of the message	Message template
Payment receiver	Protocol AND Perf=Propose
Payment sender	Protocol AND Conv-Id AND In-Reply-To AND Perf=(reject-proposal OR accept-proposal)
Payment receiver	Protocol AND Conv-Id AND In-Reply-To AND Perf=(query-if OR failure)
Payment sender	Protocol AND Conv-Id AND In-Reply-To AND Perf=inform

the message that is received is in fact respecting the sequence of the protocol. The sequence of messages in the payment protocol is such that each message, apart from the first one, is a reply to the earlier message. The first message can be identified just by the performative and protocol. Performative is added to the message templates, to make sure, that the counterparty is complying with the protocol. Table 4.1 presents the message templates for each incoming message.

In JADE, ontology is the structured definition of the environment where the agents operate [59]. By sharing the same ontology, the agents can communicate in a way that they are using common terminology. Ontology can contain concepts, predicates and actions. In JADE, an ontology is practically a set of classes, that each implement an interface of either Concept, Predicate or Action. The classes are added to a common Ontology class that defines the ontology. The structure of the ontology class in JADE is relatively complex [89]. The syntax of the ontology can be simplified by defining the ontology class as BeanOntology [90]. BeanOntology requires that each class included in the ontology complies with the JavaBean class structure, which is a standardized Java class structure that has standardized get and set functions for each class variable.

A new BeanOntology is created for the protocol. The class structure of the ontology is presented as UML class diagram in Figure 4.4. A payment proposal class is an extension of the payment value class, such that payment proposal also includes the payment identifier. The action *AcceptPaymentProposalAndCreateLNInvoice* is an action that is sent in the initiating message, in which the sender of the payment proposes to make the payment. By sending the action, the initiator instructs the receiver of the payment to accept the proposal and create a LN invoice. The action contains the payment proposal, so that the receiver knows what it is that they are accepting. The predicate *PaymentProposalAccepted* contains the payment proposal as well as the LN invoice, since the invoice is created at the time when the proposal is accepted. If the payment proposal is rejected the PaymentProposalAccepted predicate is enclosed in a Not predicate, to signify that the proposal is not accepted.

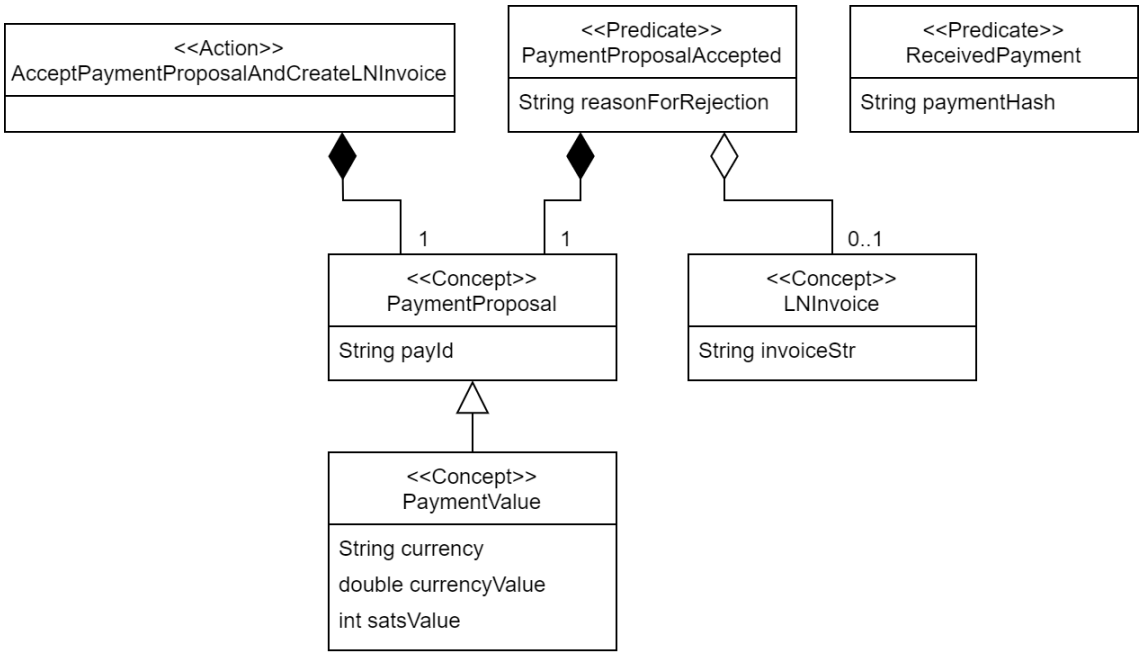

Figure 4.4. *The ontology presented as a class diagram.*

Once the sender of the payment has made the payment, they send the *ReceivedPayment* as a Query-If message, to inquire if the payment is received. If the receiver agrees to have received the payment, they respond with the RecievedPayment predicate. If they instead disagree, they respond with a Not predicate enclosing the ReceivedPayment to signify that they have not received the payment. This usage of predicates and actions is the FIPA-compliant communication approach.

The structure of the message payload is defined by the content language. FIPA-SL is the content language that is most commonly used in FIPA-compliant MASs. It is a logical language consisting of predicates, actions and terms. The language is human readable as long as the basics of the syntax is known. This eases the debugging of the protocol. The SL language is chosen as the content language of the implementation. JADE makes it possible to read the SL language and convert the expressions into Java objects.

The internal behaviour of the base agents is based on their internal state. The state is defined as an Enum variable. The states of the payment sender agent are: Initial, Proposed, Paid, Success and Failure. The states of the payment receiver agent are: Initial, Proposal Accepted, Success and Failure. When the payment behaviour of an agent is active, it is in one of these states. The behaviours start from the Initial state. As the execution of the protocol progresses the state changes. The state defines which block of code to execute when a message corresponding to the current message template arrives. The behaviours conclude when the state is either Success or Failure.

To facilitate the different use cases, two functions are defined that can be overridden in the use case specific classes. As stated in Chapter 3, an ability for the receiver to decline

a payment is an important aspect for some use cases. This functionality is facilitated by adding a function for the payment receiver agent that determines if the proposal is accepted. The function is executed after the receiver of the payment receives the initiation message, containing the payment proposal. By overriding this function in the use case specific agent, the receiver can define on what conditions is the payment accepted. Another function that can be overridden is the function that defines what happens after the payment is complete. Both of the agents contain this function. The function is always executed as the last action before concluding the payment behaviours. Both functions are run with parameters that contain information about the current circumstances. For example, the function which runs when the payment is finished, has a parameter that contains information of the payment that was done, or alternatively information that the payment failed. Likewise, the function that defines if a payment proposal is accepted, provides information about the payment proposal.

4.3 Interfaces

There are two APIs that the agents must interact with: the price API and the LN node API. This section expands on the agent-resource interactions of the methodology [59] used in Chapter 3 by outlining how the API components are implemented in the agents. As stated in Chapter 3, the agents communicate with the APIs directly, so no additional agents are needed. The components interacting with the APIs are classes of which instances are constructed in the agent classes. This is shown in the class diagram in Figure 4.3. The class communicating with the price API is the *PriceAPI* class and the class communicating with the LN node is the *LNgrpcClient* class.

The PriceAPI class can actually be any class that implements a PriceAPI interface that is show in Program 4.1. This structure enforces the dependency inversion principle and the Liskov substitution principle of the SOLID priciples. As shown in Figure 4.3, the PriceAPI class is in fact embedded in the *PriceAPIWrapper* class. The purpose of the wrapper class is to return the amount of Satoshis corresponding to an amount in a traditional currency. It uses the bitcoin price retuned by the PriceAPI to calculate the correct amount. The wrapper class and the price api classes are separated to keep the PriceAPI interface as simple as possible for it to be very easily implementable. An instance of the PriceAPI is instantiated in the use case specific agent class and given as a parameter to the *setPriceAPI()* function. This way the different price APIs can be dynamically defined for different use cases.

```
public interface PriceAPI {
    double getBTCPrice(String currency);
}
```

Program 4.1. *The PriceAPI interface.*

The PriceAPI interface is very simple and does not restrict where the price of bitcoin is sourced. For testing purposes, two separate implementations of the PriceAPI are implemented. Both of them are public Representational State Transfer (REST) APIs. The first one is offered by CoinGecko [91] and the second one by CoinDesk [92]. For each API, a REST client was implemented in a class that implements the PriceAPI interface. By using different APIs for each agent, a realistic situation is simulated, where the agents might have a contradicting knowledge of the bitcoin price.

If the system is used in a demanding environment where errors and inaccuracies are wanted to be avoided as much as possible, it would be even possible to implement the PriceAPI in such a way that the price is fetched from multiple sources within the class. The correct price could then be deducted by for example returning the average or the median price of the different sources and by discarding prices that deviate too much from the average. This would ensure that the operation of the agent is more reliable, since a faulty operation of a single API would not necessarily lead to an error situation during the payment interaction.

The other type of external resource that the agents must interact with, is the LN node. The LN node client class is instantiated by calling the *setLNHost()* function in the use case specific class. The information to form a connection with the LN node is given as parameters for this function. This way connections to different nodes can be dynamically defined in the use case specific class. The class *LNgrpcClient* is implemented to communicate with the LND software. By replacing the LNgrpcClient by another implementation, other LN node implementations such as c-lightning or Eclair could be used as the LN node.

A gRPC API connection is used to connect to the LND. gRPC is an extensive API framework. It not only conducts the transfer of information, but also provides authentication, authorization and encryption of the connection. The usage is also easy, as the commands to the API resemble calling standard functions. However, setting up the client correctly is not as easy as a REST client. Fortunately, LND developers provide instructions for implementing a Java client for LND gRPC client [93]. The instructions were used to implement the gRPC client that handles the authentication of the LND instance via the .cert file and authorization using the macaroon file [79]. Because of the authorization and authentication, it would be secure to even connect to an LND instance over the public internet.

Multiple RPC commands [94] are used to in different steps of the protocol. In addition to generating the invoice and making the payment by using the invoice, multiple commands are used for different verifications as outlined in Chapter 3. As described in Chapter 3, the description field of the invoice is used to associate the agent interaction with the LN payment. It is the agent receiving the payment that generates the invoice and the

```
private String createJsonMemo(
                String convId,
                String payId,
                double amountCurr,
                String currency
        ) {

        JSONObject jsonMemo = new JSONObject();
        jsonMemo.put("convId", convId);
        jsonMemo.put("payId", payId);
        jsonMemo.put("amountCurr", amountCurr);
        jsonMemo.put("currency", currency);

        return jsonMemo.toJSONString();
}
```

Program 4.2. *The function to create the description field of the LN invoice.*

description. Before making the payment, the payer agent verifies that the description of the invoice is correct. Because of this verification, the agents must have a common notion of the exact contents of the description. The description, also called the memo, is therefore defined as a JavaScript Object Notation (JSON) string containing the keys: *convId*, *payId*, *amountCurr* and *currency*, where the values corresponding the keys are: the FIPA conversation identifier, the payment identifier, the amount of the payment in traditional currency and currency code of the traditional currency. The generation of the description text is outlined in Program 4.2.

4.4 Adaptation to a use case

To further test the implementability to a use case, the protocol is used to extend existing agents. JADE offers a variety of examples on how to use the framework [95]. One of the examples is the book trading agents. The example consists of two agents: BookSellerAgent and BookBuyerAgent. A set of books with a title and a price can be defined to be contained in a catalog of the BookSellerAgent. The books in the catalog are the books that the agent is willing to sell. A BookBuyerAgent is in turn initialized with a single book title that it aims to buy. When active, the BookBuyerAgent periodically queries all the BookSellerAgents for the book that is aims to buy. The BookSellerAgents respond if they have the book or not. Of all the books found, the BookBuyerAgent selects the one with the lowest price and initiates a book buying protocol with the seller. In this protocol, the buying of the book is simply an agreement between the agents that the book is traded. Nothing is delivered in reality. The BookSellerAgent removes the sold book from it's catalog of books that can be sold.

Since the example contains the act of buying a product that has a price defined for it, it is a well suited example for testing the LN payment protocol. The BookSellerAgent and the BookBuyerAgent are classes that are extended from the Agent class. The classes are redefined to be extended from the base classes of the LN payment protocol. The BookBuyerAgent is extended from the PaymentSenderAgent and the BookSellerAgent is extended from the PaymentReceiverAgent. setLNHost() and setPriceApi() functions are added to be called in the setup function of the book trading agents. The PaymentReceiveBehaviour is started in the setup of the BookSellerAgent. A paymentComplete() function is overridden for the agents to print information for the user of the completed payment.

The functionality of the BookBuyerAgent is modified such that at the moment in the original BookBuyerAgent when the book is successfully purchased, the LN payment protocol is initiated instead. The price for a book in the original book trading example does not contain a unit or a currency, but it is defined to be a plain integer value. The LN payment protocol requires a currency for the payments. The unit of the book price is chosen to be euro cents to allow the integer values to be of small values. The payment identifier of the payment protocol need to be defined to link the payment to the book that is paid for. In the original book trading example, the buyer sets a unique reply-with parameter for the message where it places the purchase order for a book. The unique reply-with parameter is in the format of a word “order” concatenated with the current unix timestamp. This parameter can be called the order identifier. To identify the book as well as to ensure the uniqueness of the payment identifier, the payment identifier is constructed to contain both, the book title and the order identifier. This way the protocol manages to trade multiple books with the same title and a payment is unequivocally associated with a single order.

In this use case it is important for the seller to verify that the buyer of the book pays a correct price for the book. The implementation of the original BookSellerAgent is modified such that at the moment when the order of a book is accepted, instead of removing the book altogether, it is moved to a separate catalog of the books that are pending payment. The order identifier is included within this new catalog. The isProposalAccepted() function is overridden to verify that the payment is for one of the books that is agreed to be purchased and thus contained within the catalog of books that are pending payment. The price, the order identifier and the title of the book are verified to be correct before accepting a payment. If they correspond to the values in the payment proposal, the payment is accepted and the agents proceed to fulfill the payment. Finally in the paymentComplete() function that is executed as the payment is done; the seller removes the book from the catalog of books that are pending payment to signify that the book is bought and paid for.

The complete interaction of the agents is presented in Figure 4.5. The figure is an output of the JADE sniffer tool that is used to inspect the messages sent in a JADE system. The arrows represent messages between the agents. The performative of each message

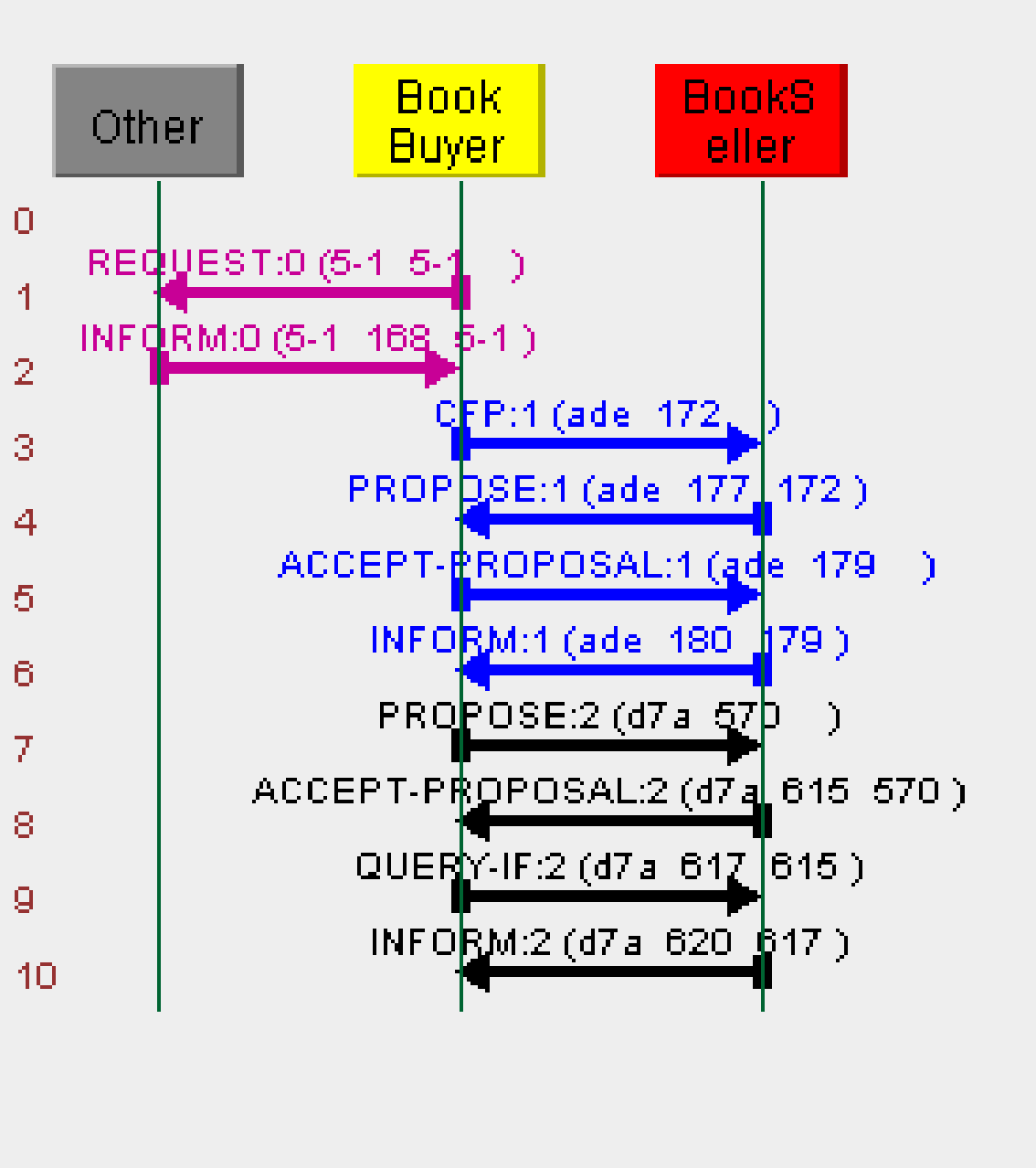

Figure 4.5. *The interaction between the book trading agents.*

is written on top of the message. The different colors represent different protocols via which the agents interact. The messages colored blue are the messages of the book trading protocol provided in the original JADE example, and the messages colored black are the messages of the LN payment protocol. Therefore, it can be concluded that the system works as intended; the agents first form the agreement that the book is traded, after which the payment for the purchase takes place. In the paymentComplete() function of each agent, appropriate variables were printed to console to further confirm that the payment was made and that the catalogs of the books behave as intended.

5. ASSESSMENT OF THE PERFORMANCE AND VIABILITY

In this chapter the design and implementation is analysed and tested to be able to answer the research questions posed in Chapter 1. First the possible error scenarios that could arise during execution of the implementation are analysed. Then the performance of the implementation is tested and analysed by making a series of payments. Finally, the viability of the protocol in real-life scenarios is discussed.

5.1 Analysis of possible error scenarios

Multiple types of error situations were identified in Chapter 3. Some of the errors can possibly be corrected automatically, while some cause the payment to fail. The most serious type of error causes the sender and the receiver of the payment to disagree on the status of the payment. In the implementation only little emphasis is put on the error situations. Therefore, they are addressed in this section. The different types of errors are not differentiated in the implementation. Only a textual description of the error is provided while the user is notified of an error. To allow for different kind of error handling for different types of errors, structured error types should be implemented.

The least serious of the error situations are the ones that can be automatically recovered from. This kind of error situation happens when the receiver of the payment rejects a payment proposal even though the proposal is of a payment that the receiver deems valid. There are several possible causes for this kind of behaviour. One possible cause is the difference between the bitcoin prices provided by the PriceApi of each agent. The receiver of the payment defines a tolerance for the price deviation that they accept, and if the calculated deviation is larger than the accepted tolerance, the receiver does not accept the payment. It is always the price proposed by the sender of the payment that is used for the payment. This is the case to assure that the payment always corresponds to the original payment proposal. The behaviour of the implementation is such that, if the receiver rejects the payment proposal, the proposal is retried three times before concluding the protocol as failed. During the retries the price of bitcoin is refetched such that the deviance in price might decrease on a new try and thus be within the tolerance. This behaviour does not guarantee optimal operation and could be improved by adding

a possibility for the sender of the payment to adjust the price in the payment proposal. This would be possible if a structure was defined for the rejection message. This would give the sender of the payment a way to react to the rejection in an optimal way instead of just simply retrying. For example, the receiver could include in the rejection message the price of bitcoin that they think is correct, such that the sender of the payment could then choose to use that price if they deem it acceptable.

Another reason for why the receiver of the payment might reject the payment is if there is a misunderstanding in communication, such that the sender proposes a payment with a wrong payment identifier, wrong price or wrong currency. This kind of failure is harder to fix by retrying the payment, but the structured rejection message could be useful also in this case. If the structured rejection was defined to contain this kind of error, the sender of the payment could alert the entity owning the agent, such that the miscommunication could be fixed for future use. An error at the proposal stage could also be caused by a lacking incoming LN balance for the receiver. Before accepting the payment, the receiver queries their maximum incoming channel balance that sets the upper limit for the incoming payments. If the incoming balance is smaller than the proposed payment, the receiver rejects the payment since it is theoretically impossible to be received. This type of error cannot be solved automatically, as it requires active management of LND payment channels. A last type of error during the proposal stage is a technical error, for example an error communicating with the price API or the LND node. In some cases, this could be solved by simply retrying the payment.

After the proposal is accepted, the sender attempts to make the payment. Several possible error scenarios are identified at this stage of the protocol. The automatic error recovery is not anymore possible at this point. If a failure happens at this point, the protocol concludes and both of the agents regard the payment as not completed. At this point the receiver of the payment has constructed an LN invoice that the sender uses to pay the invoice. If the receiver does not follow the protocol and generates an invoice that for example has invalid description field, the sender refuses to accept the invoice and the protocol ends. This should not happen, if a correct implementation is used. Another possible error scenario happens when the payment itself fails. This is expected to be a relatively common error. This is caused by complications of the LN node. For example, there might not be a valid payment route to be found in the LN, and therefore a payment cannot be made. To mitigate this type of error, the user should make sure that their LN node has sufficient balance and payment channels established. The payment channels can even be explicitly created in a way that ensures a reliable payment route between the counterparties.

The errors that happen after the payment is made are very critical. Errors at this point cause the agents to disagree if the payment is made. This is obviously critical, since the sender of the payment has then essentially lost the funds that they sent, but the receiver

Table 5.1. *The errors that might occur during the protocol.*

Stage of the protocol	Type of error	Reason for the error
Initial proposal sent	Recoverable automatically	Differing notions on the price of bitcoin
Initial proposal sent	Possibly recoverable by improving the implementation	Miscommunication prior to initiation of the protocol.
Initial proposal sent	Conclusive	Insufficient incoming balance on the payment channels of the receiver.
Proposal accepted	Conclusive	Error in API communication.
Proposal accepted	Conclusive	One of the agents not complying with the protocol, because of a faulty implementation.
Proposal accepted	Conclusive	Error in making the payment.
Payment made	Conflicting	Error in API communication.
Payment made	Conflicting	One of the agents not complying with the protocol, because of a faulty implementation.
Any	Indefinite wait for reply	Communication between agents is aborted.

refuses to deliver the product or service that was paid for. The design of LN assures that the payment is fully delivered as soon as the one making the payment receives a confirmation of the payment. Therefore, this type of error should be very rare. The only reasons that are identified for this kind of error to happen are either a technical error or one of the agents being a malicious actor. The technical error could for example be that the receiver of the payment loses their connection to the LN node. In that case they cannot confirm that the payment is made, and they consider the payment failed. If either of the counterparties is malicious, they can change the implementation in such a way that they provide false information intentionally.

In all of the error cases mentioned the agents conclude the protocol and notify the user of the payment. However, there is one type of error that is not currently handled by the implementation. If a connection between the agents is lost or either of the agents is destroyed during the interaction, the other agent might be left waiting for a reply message indefinitely. To handle this, an error on timeout should be implemented.

All the errors that were identified are collected in Table 5.1. Each identified cause for an error is listed with the type of error that it causes. Also, the stage of the protocol is displayed as similar errors at different stages of the protocol might cause different types of errors. As mentioned in the beginning of this section, the types of errors are currently not

explicitly distinguished in the implementation. In the table, the error type of *recoverable automatically*, means that the protocol might recover from the error automatically. If the recovery does not succeed, this type of error turns into a *conclusive* error. The *conclusive* type of error indicates an error that cannot be recovered from, but where the counterparties agree that the payment was not made. A *conflicting* type of error is the error where the agents disagree on the status of the payment.

5.2 Inspected performance of the system

To answer the research question “Is Bitcoin Lightning Network a valid technology to be used for agent to agent micro-payments?”, the performance of the system was tested by sending a series of payments between the agents. A total of 15 payments were sent. The tests were done in the batches of 5 payments. In each batch, the 5 payments were made with a 40 second interval. In the first batch the amount of each payment was 0.01 euro, in the second batch 0.1 euro and in the final batch 1 euro. The payments were made at a time when the price of bitcoin was relatively stable. The configuration of the LN nodes was the one that is explained in Chapter 4. The LN nodes used by the agents were connected to separate public nodes. By examining the payments directly using the lncli tool, it was confirmed that the route of each payment was three hops long, thus conveyed directly from between the two public nodes that the nodes were connected to. Statistics about the payments is collected in Table 5.2.

The table contains the amount in euros that the agent was set to pay, as well as the amount in satoshis that was actually paid via LN. A timer was implemented in the code to track how long does it take to make the payment. Two separate times were measured. The first one is the time for the whole interaction. It is the time from the moment when the payment behaviour is started for the agent sending the payment to the moment when the agent receives the last message of the protocol. The other time that was measured is the time that the LN node sending the payment takes to make the payment. This time is the time it takes for the RPC command for making the payment. The fee that the LN charges to make the payment is also included in the table, as well as the price deviation between amounts that the agents regard as the correct value for the payment. Each agent calculates amount of satoshis corresponding the amount in euros using their PriceApi. These values are used to calculate the price deviation. As he values are rounded to an even number of satoshis, the deviance does not exactly equate to the difference of the bitcoin prices indicated by the PriceApis.

In LN, the amounts are often represented in satoshis. However, to send payments smaller than one satoshi or to collect fees for less than a satoshi LN has also capability to manage the amounts in millisatoshis. One satoshi equals thousand millisatoshis. By inspecting the payments using the lncli tool, it was noted that fees for the payments were denominated

Table 5.2. *Series of payments made between the agents.*

N	Amount (€)	Amount (sat)	Time for the whole interaction (ms)	Time for LN payment (ms)	Fees (msat)	Price deviation in sat values (%)
1	0.01	19	4592	3032	1002	0
2	0.01	19	9241	8096	1002	0
3	0.01	19	3555	2815	1002	0
4	0.01	19	3097	2746	1002	0
5	0.01	19	5929	3598	1002	0
6	0.1	190	9914	7254	1019	0
7	0.1	190	3575	3122	1019	0
8	0.1	190	3595	2845	1019	0
9	0.1	190	3054	2690	1019	0
10	0.1	190	8788	7816	1019	0
11	1	1902	28764	27415	1191	0.264
12	1	1902	7394	6152	1191	0.264
13	1	1902	10505	9653	1191	0.264
14	1	1905	17566	17093	1191	0.422
15	1	1904	6599	3331	1191	0.316

in millisatoshis. The fees were relative to the payment. The fee consists of a sum of the fees collected by the intermediary nodes. In this case one of the nodes collected a static 1 sat fee, while the other used a fee that was relative to the size of the payment. The relative fee appears to be 0.01 % of the payment amount. The fees for each payment is presented in Table 5.2.

The fee for each of the payments was around one satoshi. This corresponds to around 0.0005 euros, or 0.05 cents, at the time of writing. As an absolute number, the fee is very small, but when comparing it to the smallest of the tested payments it is not negligible. It was around 5.3 % of the payment. For the other payments the fee was around 0.5 % and 0.05 % respectively. However, if both of the nodes were connected to the node that charged the relative fee of 0.01 %, the fee would have been the 0.01 %, which is negligible for micropayments.

The times that it took to make the payments was longer than expected. The fastest time to conclude the protocol was around 3 seconds, while the longest time was almost 30 seconds. As seen from the Table 5.2, the time to make the actual payment over the LN takes majority of the time in all of the payments that were made. By subtracting the time for the payment from the time for the whole protocol, the time for the protocol, excluding the time for the payment can be concluded. The shortest time for the protocol excluding

the payment was 0.35 seconds, while the longest time was 3.27 seconds. It is assumed that most this time is spent on waiting the calls to the external resources such as the calls to the PriceApi as well as the calls to generate the invoice on the LN node and querying the status of the payment from the LN node.

There are no underlying constraints on the speed of LN payments. They are only constrained by the processing power and networking efficiency of the nodes. The Raspberry Pi model 3 that was used to host both of the LN nodes simultaneously is a device with limited computational strength. Therefore, it can be assumed that using stronger hardware to host the nodes the speed of the payments would improve. Indeed, the time to make a payment over three hops was measured in a paper from 2019 [96]. The measured maximum time for a payment was 5.4 seconds, while the average time was 2.9 seconds. The hardware used in the measurement is unknown, but since the measurement is around 2 years old, the LN software implementations might have been updated to be more efficient since then. Therefore, by upgrading the hardware of the LN node, the execution time for the protocol could be made significantly faster.

Another factor that could delay the protocol might be the PriceApi as it is a request to an external web resource. To mitigate this delay, it would be possible to create a buffer for the price of bitcoin by fetching the price periodically to always keep a recent price stored locally, so fetching the price during the protocol execution would not cause any delay, as the stored price could be provided without initiating a separate web request.

Assumably, by upgrading the hardware of the LN nodes and implementing the price buffer functionality, it would be possible to decrease the time for the protocol to be around 5 seconds or even faster. To further optimize the time for the payment, it could be possible choose the routing nodes to be nodes that are known to be highly responsive.

The price deviation for all the payments of 0.01 and 0.1 euros was zero. This is because the prices provided by the PriceApis rounded to satoshis yielded the same value. However, in a case when the price of bitcoin moves to a certain value where one of the PriceApis would round the value of 0.01 euros to 19 satoshis and the other to 20 satoshis, the deviation would be around 5 %. For many cases this deviation would not be acceptable anymore. To mitigate this issue of high probability of relatively high calculated price deviation at certain price levels, the protocol should be transformed to use millisatoshis as the base unit. This would also provide more accurate payment amounts for very small payments.

The payments of 1 euro had a price deviations between 0.26 % and 0.32 %. The price deviations varied, because between the payments, the price of bitcoin fluctuated, and one of the PriceApis was more robust than the other in reflecting the price movements. The deviations of less than 0.5 % are assumed to be acceptable for most use cases.

5.3 Viability for business use

Even though the fees for the payments in LN can be very affordable compared to other payment methods, the speed of the payments is acceptable for many use cases and a suitable protocol can be devised, there are still multiple issues on why a business might not want to use the LN as a micropayment infrastructure. One notable issue in adopting a Bitcoin based payment infrastructure is the need for owning bitcoin, and thus accepting the risk of the value of the held assets fluctuating.

One solution to the aforementioned problem of high volatility is a construct called Discreet Log Contract (DLC) [97][98]. DLC is a second layer construct that brings smart contracting capabilities to Bitcoin such that information external to the blockchain can be used to affect the outcomes of transactions happening on-chain. This is done using oracles and the multisignature transactions as in LN. Oracles are third party entities that bring external data to the blockchain [99]. Many different types of implementations of oracles exist in multiple blockchains that have smart contracting capabilities. The operation of a DLC is such that two parties that own bitcoin, can lock their funds in a common contract such that at the time of locking the funds, they decide a set of future outcomes of some external information that correspond to different amounts of funds to be later distributed to each party. The external information can for example be the price of bitcoin. The two parties can for example create the contract such that at a certain date in future, the other party receives the amount of funds that correspond to the amount in traditional currency at the time when the contract was created. This way one of the parties of the contract takes on all the risk and the possible reward of the other party. By using this type of construct, a business could hold bitcoin without being exposed to the market risk. There exists research on integrating the DLCs to the LN [100], which would be required to mitigate the market risk of bitcoin held within the LN. Another possibility to mitigate the market risk would be to use a custodian service that, for a fee, manages the LN node as well as stabilizes the amount of bitcoin held depending on the price movements. Most businesses would probably be reluctant in holding bitcoin, unless the price of bitcoin stabilizes in the future, or accessible tools for mitigating the market risk are made available.

One problem that was identified is the LN topology: if the different parties adopting the protocol randomly choose the nodes they form the payment channels with, there is no guarantee that there exists a route to make the payments with low fees. This problem could be mitigated by a relatively simple solution. The entities using the protocol could decide or even create a central hub node that the payments are routed through. This would assure that a short route with low fees exists for the payments. This solution somewhat contradicts with the principle of decentralization of the LN. The tendency of LN to form a topology where the network is concentrated around only a handful of central hubs is criticized in a paper by Erdin et al. [101]. The authors propose that instead of LN, other

private, Bitcoin based, payment channel networks (PCNs) could be formed for different use cases. The proposed PCN structure would have a more balanced topology and stronger guarantees of payments finding a route. This solution could be an alternative way of forming the payment infrastructure of the protocol instead of LN.

If Bitcoin and LN continue to gain adoption and the software, topology, tooling and capacity around LN improves, much of the problems described in this section would be eased.

6. CONCLUSION

This final chapter concludes the research. The research questions are answered by referring to the findings of the previous chapter. Furthermore, improvements to the implementation and topics for further research are proposed, based on the issues that were identified during the research.

6.1 Conclusions

The main objective of the thesis was to design, implement and analyse a proof of concept protocol for M2M micropayments using the Bitcoin LN and FIPA multi-agent framework. A viable protocol that fulfills the functional and non-functional requirements set in Chapter 3 was successfully designed and implemented. The protocol is capable of sending payments between FIPA agents by linking the agents to LN nodes. The payments are denominated in desired traditional currency and verified by both parties of the transaction. The implementation was made in a way that it is use case agnostic and the implementation can be easily adapted to multitude of use cases.

Minimal transaction fees and times were set as objectives for the implementation. The desired time to make a payment was less than 1 second and the desired fee for a single payment was 0.01 euros. As the implementation was tested, the observed time for a payment was between 3 seconds and 30 seconds depending on the monetary amount that was sent. For payments of 0.1 euros or less the time was always less than 10 seconds. Therefore, it can be concluded that the implementation did not meet the desired speed for payments. The reason for the inadequate speed of the payments is assumed to be mostly attributable to the constrained hardware that was used to operate the LN node software. By upgrading the hardware, the speed for the payments could significantly increase. The fee for each payment was around 1 satoshi, which corresponds to 0.0005 euros. Therefore, regarding fees, the objective of low fees was met as the fees were more affordable than expected.

Even though a functional protocol with desired functionalities was devised, some obstacles were identified on why it would be difficult for a business to adopt this kind of protocol for M2M micropayments. The most significant problem that was identified was the need to hold bitcoin to be able to make the payments, and thus be exposed to its market volatility

as an asset. If a solution for this problem is found and the popularity of Bitcoin continues increasing, some type of standardized LN micropayment protocol could be a viable approach for M2M micropayments in the future.

6.2 Future work

The implementation is only developed to act as a proof of concept. Therefore, not all the features that are desired for a system used in full production usage are implemented. As described in Chapter 5 multiple features that are important for handling the error scenarios could be implemented. One notable improvement would be to define a structure for error messages, such that the agent that is notified of an error, could react each type of error in a distinct way. Another essential improvement in error handling would be include error handling for timeout scenarios where one of the agents becomes unresponsive.

Despite error handling, multiple possible improvements were identified during the analysis phase in Chapter 5. To allow for more accurate payments and better behaviour for very small payments, the base unit of the protocol could be changed to millisatoshis instead of satoshis. To improve alleviate the problem of slow payments, the LN nodes should be tested on stronger hardware. Also, a price buffer functionality could be implemented, such that it would not be necessary to delay the protocol by fetching the price of bitcoin each time during the protocol execution.

The protocol was strived to be designed in a way that it is implementation-agnostic, meaning that agents with new, but protocol compliant, implementation should be interoperable with the original implementation. This is however not tested as only one implementation of each agent is created. To verify the integrity of the protocol, the protocol should be tested using independently developed implementations. It is also not tested if the current implementation can handle multiple payments executing in parallel.

A further research topic would be related to the LN side of the payment system. Issues to research on that area include implementing the DLC functionality to mitigate the market risk of holding bitcoin. Another topic is to determine if it is possible to leverage the research by Kurt et al. [9] mentioned in Chapter 2 to use a third party gateway LN node to make the payments. This way the entity controlling the agent would not need to take care of the LN infrastructure themselves. Alternatively, a topic for research would be to determine the optimal LN node configuration and methodology for opening and managing payment channels.

REFERENCES

[1] Robert, J., Kubler, S. and Ghatpande, S. Enhanced Lightning Network (off-chain)-based micropayment in IoT ecosystems. *Future Generation Computer Systems* 112 (2020), pp. 283–296. ISSN: 0167-739X. DOI: 10.1016/j.future.2020.05.033.

[2] Mercan, S., Erdin, E. and Akkaya, K. Improving transaction success rate in cryptocurrency payment channel networks. *Computer Communications* 166 (2021), pp. 196–207. ISSN: 0140-3664. DOI: 10.1016/j.comcom.2020.12.009.

[3] Nakada, R., Nguyen, K. and Sekiya, H. Implementation of Micropayment System Using IoT Devices. *Journal of Signal Processing* 25.4 (2021), pp. 137–140. DOI: 10.2299/jsp.25.137.

[4] Strugar, D., Hussain, R., Mazzara, M., Rivera, V., Young Lee, J. and Mustafin, R. On M2M Micropayments: A Case Study of Electric Autonomous Vehicles. eng. *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)*. IEEE, 2018, pp. 1697–1700. DOI: 10.1109/Cybermatics_2018.2018.00283.

[5] Ince, D. *A dictionary of the Internet*. eng. 4th ed. Oxford University Press, 2019. ISBN: 9780191884276. DOI: 10.1093/acref/9780191884276.001.0001.

[6] *What are micropayments?* PayPal. URL: https://www.paypal.com/us/smarthelp/article/what-are-micropayments-faq664 (visited on 06/02/2021).

[7] Zhang, Y. and Wen, J. The IoT electric business model: Using blockchain technology for the internet of things. English. *Peer-To-Peer Networking and Applications* 10.4 (July 2017), pp. 983–994. DOI: 10.1007/s12083-016-0456-1.

[8] Ensor, A., Schefer-Wenzl, S. and Miladinovic, I. Blockchains for IoT Payments: A Survey. Dec. 2018, pp. 1–6. DOI: 10.1109/GLOCOMW.2018.8644522.

[9] Kurt, A., Mercan, S., Erdin, E. and Akkaya, K. Enabling Micro-payments on IoT Devices using Bitcoin Lightning Network. (Dec. 2020). URL: https://www.researchgate.net/publication/347534420_Enabling_Micro-payments_on_IoT_Devices_using_Bitcoin_Lightning_Network.

[10] Xu, A., Li, M., Huang, X., Xue, N., Zhang, J. and Sheng, Q. A Blockchain Based Micro Payment System for Smart Devices. Aug. 2016. URL: https://www.researchgate.net/publication/311789642_A_Blockchain_Based_Micro_Payment_System_for_Smart_Devices.

[11] Nakamoto, S. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Oct. 2008. URL: https://bitcoin.org/bitcoin.pdf.

[12] Wüst, K. and Gervais, A. Do you Need a Blockchain?: *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. 2018, pp. 45–54. DOI: 10.1109/CVCBT.2018.00011.

[13] Kuiper, C. and Neureuter, J. *BITCOIN FIRST: Why investors need to consider bitcoin separately from other digital assets*. Tech. rep. Fidelity Digital Assets, Jan. 2022. URL: https://www.fidelitydigitalassets.com/bin-public/060_www_fidelity_com/documents/FDAS/bitcoin-first.pdf.

[14] *Hasn't Bitcoin been hacked in the past?* bitcoin.org. 2021. URL: https://bitcoin.org/en/faq#hasnt-bitcoin-been-hacked-in-the-past (visited on 01/08/2022).

[15] Parkin, J. The senatorial governance of Bitcoin: making (de)centralized money. *Economy and Society* 48.4 (2019), pp. 463–487. DOI: 10.1080/03085147.2019.1678262.

[16] *The 3 Phases of Ethereum's 2.0 Serenity Upgrade*. Gemini. 2021. URL: https://www.gemini.com/cryptopedia/ethereum-2-0-blockchain-roadmap-proof-of-stake-pos (visited on 01/08/2022).

[17] *Bitcoin Average Transaction Fee*. YCharts. 2019. URL: https://ycharts.com/indicators/bitcoin_average_transaction_fee (visited on 06/03/2021).

[18] Poon, J. and Dryja, T. *The Bitcoin Lightning Network: Scalable off-chain instant payments, Technical Report (draft)*. Jan. 2016. URL: http://lightning.network/docs/.

[19] Böhme, R., Christin, N., Edelman, B. and Moore, T. Bitcoin: Economics, Technology, and Governance. eng. *The Journal of economic perspectives* 29.2 (2015), pp. 213–238. ISSN: 0895-3309.

[20] Ammous, S. *The bitcoin standard : the decentralized alternative to central banking*. eng. 1st edition. Hoboken, New Jersey: Wiley, 2018 - 2018. ISBN: 1-119-47389-6.

[21] Ventures, V. *How And Why Bitcoin Could Become A New Reserve Currency*. 2021. URL: https://seekingalpha.com/article/4400007-how-and-why-bitcoin-become-new-reserve-currency (visited on 09/10/2021).

[22] Sharma, R. *Will bitcoin end the dollar's reign?* 2020. URL: https://www.ft.com/content/ea33b688-12e0-459c-80c5-2efba58e6f1a (visited on 09/10/2021).

[23] Taskinsoy, J. Bitcoin: A New Digital Gold Standard in the 21st Century?: *SSRN Electronic Journal* (Oct. 2021), pp. 1–59. DOI: 10.2139/ssrn.3941857.

[24] *El Salvador, primer país del mundo en reconocer al Bitcoin como moneda de curso legal*. Asamblea Legislativa El Salvador. June 2021. URL: https://www.asamblea.gob.sv/node/11282 (visited on 01/28/2022).

[25] Elmandjra, Y. *ARK Invest: Big Ideas 2022*. Tech. rep. Jan. 2022. URL: https://research.ark-invest.com/hubfs/1_Download_Files_ARK-Invest/White_Papers/ARK_BigIdeas2022.pdf.

[26] Weber, W. E. *A Bitcoin standard: Lessons from the gold standard*. eng. Bank of Canada Staff Working Paper 2016-14. Ottawa, 2016. URL: http://hdl.handle.net/10419/148121.

[27] Hanley, B. P. The False Premises and Promises of Bitcoin. *CoRR* abs/1312.2048 (2013). arXiv: 1312.2048. URL: http://arxiv.org/abs/1312.2048.

[28] Paul, A. Bitcoin vs. Sovereign Money. On the Lure and Limits of Monetary Reforms. *Behemoth: A Journal on Civilisation* 9.2 (2016), pp. 8–21. DOI: 10.5451/unibas-ep52097.

[29] *CM BTC Price*. Coin Metrics. 2022. URL: https://cmtv.coinmetrics.io/ (visited on 02/02/2022).

[30] *Bitcoin (BTC)*. CoinGecko. URL: https://www.coingecko.com/en/coins/bitcoin (visited on 02/02/2022).

[31] *Gold's Market Cap*. companiesmarketcap.com. URL: https://companiesmarketcap.com/gold/marketcap/ (visited on 02/02/2022).

[32] Wooldridge, M. and Jennings, N. R. Intelligent agents: theory and practice. eng. *Knowledge engineering review* 10.2 (1995), pp. 115–152. ISSN: 0269-8889.

[33] Poslad, S. Specifying protocols for multi-agent systems interaction. eng. *ACM transactions on autonomous and adaptive systems* 2.4 (2007), 15–es. ISSN: 1556-4665.

[34] *Foundation for Intelligent Physical Agents*. Foundation for Intelligent Physical Agents. URL: http://www.fipa.org/ (visited on 07/15/2021).

[35] Dorri, A., Kanhere, S. S. and Jurdak, R. Multi-Agent Systems: A Survey. *IEEE Access* 6 (2018), pp. 28573–28593. DOI: 10.1109/ACCESS.2018.2831228.

[36] Odell, J. J., Van Dyke Parunak, H. and Bauer, B. Representing Agent Interaction Protocols in UML. eng. *Agent-Oriented Software Engineering*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 121–140. ISBN: 9783540415947.

[37] Debnath, S., Chattopadhyay, A. and Dutta, S. Brief review on journey of secured hash algorithms. *2017 4th International Conference on Opto-Electronics and Applied Optics (Optronix)*. 2017, pp. 1–5. DOI: 10.1109/OPTRONIX.2017.8349971.

[38] *Difficulty*. Bitcoin Wiki. 2010. URL: https://en.bitcoin.it/wiki/Difficulty (visited on 06/10/2021).

[39] Brakmić, H. *Bitcoin and Lightning Network on Raspberry Pi Running Nodes on Pi3, Pi4 and Pi Zero*. eng. 1st ed. 2019. Berkeley, CA: Apress, 2019. ISBN: 1-4842-5522-4.

[40] Park, S., Im, S., Seol, Y. and Paek, J. Nodes in the Bitcoin Network: Comparative Measurement Study and Survey. *IEEE Access* 7 (2019), pp. 57009–57022. DOI: 10.1109/ACCESS.2019.2914098.

[41] *Script*. Bitcoin Wiki. 2010. URL: https://en.bitcoin.it/wiki/Script (visited on 06/10/2021).

[42] *Transactions*. Bitcoin Wiki. 2010. URL: https://developer.bitcoin.org/devguide/transactions.html (visited on 06/11/2021).

[43] *P2P Network*. Bitcoin Wiki. 2010. URL: https://developer.bitcoin.org/devguide/p2p_network.html (visited on 06/11/2021).

[44] *Mining*. Bitcoin Wiki. 2010. URL: https://developer.bitcoin.org/devguide/mining.html (visited on 06/11/2021).

[45] *About*. Bitcoin Core. 2021. URL: https://bitcoincore.org/en/about (visited on 07/14/2021).

[46] *bitcoin/CONTRIBUTING.md*. Bitcoin Core. 2021. URL: https://github.com/bitcoin/bitcoin/blob/master/CONTRIBUTING.md (visited on 07/14/2021).

[47] *Network Snapshot*. BitNodes. 2021. URL: https://bitnodes.io/nodes (visited on 07/14/2021).

[48] Antonopoulos, A. M., Osuntokun, O. and Pickhardt, R. *Mastering the Lightning Network*. eng. O'Reilly Media, Inc, 2021. ISBN: 9781492054856. URL: https://github.com/lnbook/lnbook.

[49] *lightningnetwork/lightning-rfc*. Lightning Network. 2021. URL: https://github.com/lightningnetwork/lightning-rfc (visited on 07/14/2021).

[50] Jian-Hong, L., Primicerio, K., Squartini, T., Decker, C. and Tessone, C. J. Lightning network: a second path towards centralisation of the Bitcoin economy. English. *New Journal of Physics* 22.8 (Aug. 2020). DOI: 10.1088/1367-2630/aba062.

[51] Bertucci, L. Incentives on the Lightning Network : A Blockchain-based Payment Network. eng. *SSRN Electronic Journal* (2020). DOI: 10.2139/ssrn.3540581.

[52] Guo, Y., Tong, J. and Feng, C. A Measurement Study of Bitcoin Lightning Network. eng. *2019 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 2019, pp. 202–211. ISBN: 9781728146935. DOI: 10.1109/Blockchain.2019.00034.

[53] *lightningnetwork/lightning-rfc/11-payment-encoding.md*. Lightning Network. 2021. URL: https://github.com/lightningnetwork/lightning-rfc/blob/master/11-payment-encoding.md (visited on 07/14/2021).

[54] Russell, R. *Offers: Lightning's Native Experience, Everywhere*. 2021. URL: https://bolt12.org/ (visited on 07/14/2021).

[55] Russell, R. *Pull request: Offers #798*. 2021. URL: https://github.com/lightningnetwork/lightning-rfc/pull/798 (visited on 07/14/2021).

[56] *What Is IOTA*. IOTA Foundation. 2021. URL: https://www.iota.org/get-started/what-is-iota (visited on 01/08/2022).

[57] Boudriga, N. et al. A resilient micro-payment infrastructure: an approach based on blockchain technology. *Kuwait Journal of Science* 49.1 (2022). DOI: 10.48129/kjs.v49i1.10578.

[58] *The bIoTope Project*. The bIoTope Project. 2022. URL: https://biotope-project.eu/ (visited on 01/28/2022).

[59] Nikraz, M., Caire, G. and Bahri, P. A methodology for the development of multi-agent systems using the JADE platform. *Comput. Syst. Sci. Eng.* 21 (Mar. 2006). URL: https://www.researchgate.net/publication/220403984_A_methodology_for_the_development_of_multi-agent_systems_using_the_JADE_platform.

[60] Bauer, B., Müller, J. and Odell, J. Agent UML: A Formalism for Specifying Multiagent Software Systems. Vol. 11. June 2001, pp. 109–120. ISBN: 978-3-540-41594-7. DOI: 10.1007/3-540-44564-1_6.

[61] *FIPA communicative act library specification*. Foundation for Intelligent Physical Agents, Dec. 2002. URL: http://www.fipa.org/specs/fipa00037/SC00037J.pdf (visited on 05/26/2021).

[62] *FIPA Interaction Protocol specifications*. Foundation for Intelligent Physical Agents, 2002. URL: http://www.fipa.org/repository/ips.php3 (visited on 05/26/2021).

[63] Huget, M.-P. Agent UML notation for multiagent system design. *IEEE Internet Computing* 8.4 (2004), pp. 63–71. DOI: 10.1109/MIC.2004.6.

[64] Bellifemine, F., Poggi, A. and Rimassa, G. Developing Multi-Agent Systems with a FIPA-compliant Agent Framework. *Softw., Pract. Exper.* 31 (Feb. 2001), pp. 103–128. DOI: 10.1002/1097-024X(200102)31:2<103::AID-SPE358>3.0.CO;2-0.

[65] *ISO 4217 CURRENCY CODES*. International Organization for Standardization. URL: https://www.iso.org/iso-4217-currency-codes.html (visited on 09/14/2021).

[66] *FIPA ACL Message Structure Specification*. Foundation for Intelligent Physical Agents, Dec. 2002. URL: http://www.fipa.org/specs/fipa00061/SC00061G.pdf (visited on 09/14/2021).

[67] Shynkevich, A. Bitcoin arbitrage. *Finance Research Letters* 40 (2021), p. 101698. ISSN: 1544-6123. DOI: 10.1016/j.frl.2020.101698.

[68] Fulgur Ventures. *An Overview of Lightning Network Implementations*. Mar. 2020. URL: https://medium.com/@fulgur.ventures/an-overview-of-lightning-network-implementations-d670255a6cfa (visited on 09/27/2021).

[69] *LND Developer Site*. Lightning Labs. 2021. URL: https://dev.lightning.community/ (visited on 09/27/2021).

[70] *About gRPC*. grpc.io. 2021. URL: https://grpc.io/about/ (visited on 09/29/2021).

[71] *Installation*. Lightning Labs. 2021. URL: https://dev.lightning.community/guides/installation/ (visited on 09/27/2021).

[72] *Bitcoin Core 0.21.1*. Bitcoin Core. 2021. URL: https://bitcoincore.org/en/releases/0.21.1/ (visited on 09/27/2021).

[73] *bitcoind*. Bitcoin Wiki. 2010. URL: https://en.bitcoin.it/wiki/Bitcoind (visited on 09/28/2021).

[74] Bitcoin developers. *Block and Transaction Broadcasting with ZeroMQ*. 2015. URL: https://github.com/bitcoin/bitcoin/blob/master/doc/zmq.md (visited on 09/28/2021).

[75] *Running A Full Node - Network Configuration*. bitcoin.org. 2021. URL: https://bitcoin.org/en/full-node#network-configuration (visited on 09/28/2021).

[76] Bosworth, A. *Run LND*. 2021. URL: https://github.com/alexbosworth/run-lnd (visited on 09/27/2021).

[77] Stadicus. *Beginner's Guide to Lightning on a Raspberry Pi*. 2021. URL: https://stadicus.github.io/RaspiBolt/ (visited on 09/27/2021).

[78] *LND Overview and Developer Guide*. Lightning Labs. 2021. URL: https://dev.lightning.community/overview/ (visited on 10/11/2021).

[79] *Macaroons*. Lightning Labs. 2021. URL: https://docs.lightning.engineering/lightning-network-tools/lnd/macaroons (visited on 10/11/2021).

[80] *Lightning Network Search and Analysis Engine*. 1ML. 2021. URL: https://1ml.com/ (visited on 10/11/2021).

[81] *Node: CoinGate*. 1ML. 2021. URL: https://1ml.com/node/0242a4ae0c5bef18048fbecf9 (visited on 10/12/2021).

[82] *Node: southxchange.com*. 1ML. 2021. URL: https://1ml.com/node/0260fab633066ed7b1 (visited on 10/12/2021).

[83] *Your Gateway to All Things Cryptocurrency*. CoinGate. 2021. URL: https://coingate.com/ (visited on 10/21/2021).

[84] *About SouthXchange*. SouthXchange. 2021. URL: https://main.southxchange.com/Home/About (visited on 10/12/2021).

[85] *How to get inbound capacity on the Lightning Network*. Lightning Labs. 2021. URL: https://docs.lightning.engineering/the-lightning-network/liquidity/how-to-get-inbound-capacity-on-the-lightning-network (visited on 10/11/2021).

[86] *Radically Simple & Powerful Bitcoin Wallet*. BlueWallet. 2021. URL: https://bluewallet.io/ (visited on 10/12/2021).

[87] Martin, R. C. *Agile software development : principles, patterns, and practices*. eng. Alan Apt series. Upper Saddle River, N.J: Pearson Education, 2003. ISBN: 0-13-597444-5.

[88] Bellifemine, F. L. *Developing multi-agent systems with JADE*. eng. Wiley Series in Agent Technology ; v.7. West Sussex, England ; John Wiley, 2007. ISBN: 1-280-83865-5.

[89] Caire, G. and Cabanillas, D. *JADE Tutorial - Application-Defined Content Languages and Ontologies*. Telecom Italia S.p.A. 2010. URL: https://jade.tilab.com/doc/tutorials/CLOntoSupport.pdf.

[90] Cancedda, P. and Caire, G. *JADE Tutorial - Creating Ontologies by Means of the Bean-Ontology Class*. Telecom Italia S.p.A. 2010. URL: https://jade.tilab.com/doc/tutorials/BeanOntologyTutorial.pdf.

[91] *Explore the API*. CoinGecko. 2021. URL: https://www.coingecko.com/en/api/documentation (visited on 10/25/2021).

[92] *CoinDesk API*. CoinDesk. 2021. URL: https://old.coindesk.com/coindesk-api (visited on 10/25/2021).

[93] *How to write a Java gRPC client for the Lightning Network Daemon*. Lightning Labs. 2021. URL: https://github.com/lightningnetwork/lnd/blob/master/docs/grpc/java.md (visited on 10/25/2021).

[94] *LND gRPC API Reference*. Lightning Labs. 2021. URL: https://api.lightning.community/#lnd-grpc-api-reference (visited on 10/25/2021).

[95] *JADE Examples*. Telecom Italia S.p.A. 2021. URL: https://jade.tilab.com/documentation/examples/ (visited on 11/07/2021).

[96] Erdin, E., Cebe, M., Akkaya, K., Solak, S., Bulut, E. and Uluagac, S. A Bitcoin payment network with reduced transaction fees and confirmation times. eng. *Computer networks (Amsterdam, Netherlands : 1999)* 172 (2020), pp. 107098–. ISSN: 1389-1286.

[97] Dryja, T. Discreet log contracts. (2017). URL: https://adiabat.github.io/dlc.pdf.

[98] *discreet log contracts*. MIT digital currency initiative. 2022. URL: https://dci.mit.edu/smart-contracts (visited on 01/28/2022).

[99] Beniiche, A. A Study of Blockchain Oracles. *CoRR* abs/2004.07140 (2020). arXiv: 2004.07140. URL: https://arxiv.org/abs/2004.07140.

[100] Kuwahara, I., Le Guilly, T. and Nakagawa, T. Discreet Log Contracts Channels and Integration in the Lightning Network. (2020). URL: https://github.com/p2pderivatives/offchain-dlc-paper/blob/master/offchaindlc.pdf.

[101] Erdin, E., Cebe, M., Akkaya, K., Bulut, E. and Uluagac, S. A scalable private Bitcoin payment channel network with privacy guarantees. *Journal of Network and Computer Applications* 180 (2021), p. 103021. ISSN: 1084-8045. DOI: 10.1016/j.jnca.2021.103021.