Payment Channel Networks with Resource-constrained Devices

Gabriel Antonio Fontes Rebello

▶ **To cite this version:**

Gabriel Antonio Fontes Rebello. Payment Channel Networks with Resource-constrained Devices. Modeling and Simulation. Sorbonne Université; Universidade federal do Rio de Janeiro, 2023. English. NNT : 2023SORUS254 . tel-04247845

HAL Id: tel-04247845
https://theses.hal.science/tel-04247845

Submitted on 18 Oct 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Sorbonne Université

Universidade Federal do Rio de Janeiro

EDITE de Paris - COPPE/UFRJ

LIP6/NPA - PEE/GTA

Payment Channel Networks with Resource-constrained Devices

Par Gabriel Antonio FONTES REBELLO

Thèse de doctorat de Informatique

Dirigée par Maria POTOP-BUTUCARU, Marcelo DIAS DE AMORIM et Luís Henrique MACIEL KOSMALSKI COSTA

Présentée et soutenue publiquement le 12 juillet 2023

Devant un jury composé de :

Prof. Yacine GHAMRI-DOUDANE, Rapporteur (La Rochelle Université)

Prof. Fabíola GONÇALVES PEREIRA GREVE, Rapporteuse (UFBA)

Prof. Augusto QUADROS TEIXEIRA, Examinateur (IMPA)

Prof. Célio Vinícius NEVES DE ALBUQUERQUE, Examinateur (UFF)

Prof. Miguel Elias MITRE CAMPISTA, Examinateur (UFRJ)

Prof. Maria POTOP-BUTUCARU, Directeure de thèse (Sorbonne Université)

Prof. Marcelo DIAS DE AMORIM, Co-directeur de thèse (CNRS)

Prof. Luís Henrique MACIEL KOSMALSKI COSTA, Directeur de thèse (UFRJ)

To my friends and family.

Thanks

First of all, I thank my parents, Gabriel and Kátia, for always encouraging and striving for my academic education. I thank my sister, Mariana, for all the years of sincere companionship. Without them, this thesis would never exist.

I thank my advisors, Luís Henrique M. K. Costa, Maria Potop-Butucaru and Marcelo Dias de Amorim for agreeing to supervise this thesis. In particular, I thank them for the great effort put into making this cotutelle possible even in the face of numerous unforeseen circumstances. I thank Prof. Otto Carlos M. B. Duarte, wherever he is, for the countless life lessons and for all the loud laughs we had.

I thank my colleagues from Grupo de Teleinformática e Automação (GTA), who were fundamental to this work and to my formation as a professional. I especially thank my friends Gustavo Camilo, Lucas Airam de Souza, Lucas Chagas, Igor Alvarenga, Igor Sanz, Martin Andreoni, Diogo Mattos, Fernando Molano, and many others with whom I had the privilege of learning daily at the lab. I thank the professors from GTA, Luís, Miguel, Rodrigo, Pedro Cruz and Pedro Velloso, for all the lessons learned. GTA is and will continue to be like a home to me.

I thank all the people who have positively impacted my personal life along this path. My companions from France, Thaís Pansani, Claudio Vasconcelos, Thaysa Oliveira, Maria Cláudia, the Gerânios group, my friends from Maison du Brésil, and my dear greek friends Frosso Papanastasiou and Marilena Moustaka. Thank you for all the intense moments and I hope you are all well. Similarly, I thank my friends from Brazil, too numerous to name, for all the good times we lived here during these years. Finally, I thank my life partner, Larissa Mello, for all the support and companionship during the most challenging stage of this thesis. Life is surely better lived with you.

I thank CNPq, CAPES, FAPESP, and FAPERJ for making this work possible and for continuously promoting high-level research in Brazil. I thank Universidade Federal do Rio de Janeiro and Sorbonne Université for agreeing to establish this cotutelle. I thank GTA and LIP6 for hosting me, respectively, during the periods I was in Brazil and France.

None of this would be possible without any of the aforementioned people. This thesis is dedicated to them and to the memory of prof. Otto Carlos M. B. Duarte.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.) and to EDITE de Paris as a partial fulfillment of the requirements for the degree of Doctor (Dr.).

PAYMENT CHANNEL NETWORKS WITH RESOURCE-CONSTRAINED DEVICES

Gabriel Antonio Fontes Rebello

July/2023

Advisors: Luís Henrique Maciel Kosmalski Costa
Maria Potop-Butucaru
Marcelo Dias de Amorim

Department: Electrical Engineering

Payment-channel networks (PCN) represent the leading solution to scale blockchain-based payments to the performance levels of centralized payment systems. However, current PCNs require nodes to stay permanently online and have enough resources to execute payment security mechanisms. Such assumptions are difficult to guarantee in battery-powered devices with intermittent connectivity patterns, such as mobile phones, smart objects, and sensors. In this thesis, we address the case of PCNs with resource-constrained devices on several fronts. First, we formalize a hybrid PCN model that considers light nodes and propose a mechanism to protect payment channels with resource-constrained devices. Our experiments show that the proposed mechanism is efficient for devices with high and medium availability in mobile broadband connections. Next, we propose PCNsim, a simulator that replicates the main functionalities of a PCN in the OMNeT++ framework. PCNsim allows researchers to experiment with payments under custom networking conditions representing resource-constrained devices' connections. PCNsim's demonstrations show that it correctly reproduces the behavior of a PCN over unreliable communication channels. Finally, we address the problem of routing payments from resource-constrained devices. We present a payment scheme that anticipates payment confirmations for time-sensitive applications and two routing algorithms that route payments considering application-specific constraints. The results show that our routing algorithms are efficient both for single-path and multi-path payments and reach their top performance when the problem's constraints are tight.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Blockchain technology has revolutionized the transfer of digital assets by moving transaction processing from a central entity to a decentralized network of validators [2, 8, 9]. However, the consensus protocols validating transactions in such networks still need to improve the performance if they are to effectively replace standard payment methods. Publishing a transaction in Bitcoin [8] takes approximately one hour, can incur over $20 fees, and spends an amount of energy equivalent to the consumption of a US average household over 50 days [10]. Bitcoin's and Ethereums' throughput of approximately 7 tx/s and 15 tx/s, respectively, are orders of magnitude smaller than the average 6,000 tx/s achieved by large credit card companies [11]. Such performance issues are known in the literature as the *blockchain scalability problem*[1], which, if solved, has the potential to enable a worldwide adoption of blockchain systems in the life of ordinary citizens.

Payment-channel networks (PCNs) represent the leading solution to solve the blockchain scalability problem regarding payments. In PCNs, two users wishing to transact continuously can transfer some of their coins to a joint address in the blockchain. This process creates a *payment channel* between them in which the locked coins can be transferred immediately. It suffices to reallocate the channel funds through off-chain private transactions. The collection of payment channels between users forms a peer-to-peer *payment channel network* in which payments can be routed like packets.

Figure 1.1 depicts an example of a PCN. In the example, Alice has a payment channel of 12 coins with Bob, of which 10 are on her side. This means she can send up to 10 coins to Bob in one or more transactions without validating the transactions in the blockchain. She can also send coins to Charlie through a multi-hop payment.

[1]Scalability here refers to Buterin's concept of scalability, i.e., the ability to process more transactions per second [12]. This contrasts with the notion of scalability in distributed computing, which is the ability to maintain performance when the number of nodes in the system increases. We discuss such notions in Section 2.2.

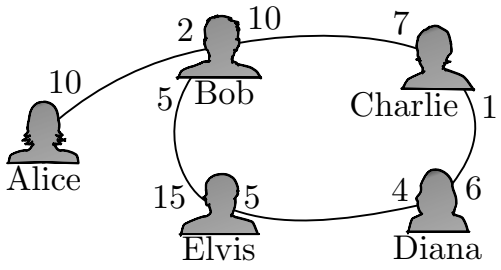

Figure 1.1: A payment-channel network (PCN) composed of bidirectional payment channels with limited capacity. Users can route payments through intermediaries to reach their destinations.

In this case, she sends the coins in her channel with Bob, and Bob relays them to Charlie. The routing process happens in a few seconds, even if Alice and Charlie do not share a direct payment channel. To ensure Bob cannot steal Alice's coins while forwarding them, a hash-based fund-locking mechanism puts Bob's coins in custody until he proves he forwarded the payment. We detail this process in Chapter 3.

PCNs reduce transaction latency from minutes to, at most, a few seconds and yield near-zero transaction costs, which significantly improves the efficiency of blockchains [13–18]. The ability to execute fast and secure payments without consensus validation narrows the gap between cryptocurrencies and real-life payments, enhancing the use of blockchain systems. For instance, PCNs have played a crucial role in the recent adoption of Bitcoin as an official currency in El Salvador [19].

1.1 Storyline and Problem Statement

Despite efficiently scaling blockchains, current PCN implementations present a significant limitation: they rely on powerful nodes to work. Specifically, they require that nodes stay online at all times and have enough resources to execute the security mechanisms involved in a payment. For example, Bitcoin's Lightning Network [2] and Ethereum's Raiden Network [20] assume that nodes maintain an updated view of the network topology, that they can verify channels in the blockchain whenever needed and that all nodes adopt onion routing [21] as the only payment forwarding mechanism to guarantee privacy [22]. This implies that nodes must have high availability to keep the topology up-to-date, enough bandwidth and storage capacity to download and maintain blocks, and strong computational power to encrypt onion packets. Furthermore, PCNs often assume connections between nodes are reliable so that payments are never dropped.

Such assumptions may be reasonable for servers or desktops but are difficult to guarantee in battery-powered devices with intermittent connectivity patterns, such as mobile phones, smart objects, and IoT sensors. These devices typically have limited

resources and communicate through lossy wireless connections that do not provide the required reliability to send payments. Furthermore, it is unrealistic to assume light nodes[2] will maintain updated topologies or execute security mechanisms that spend most of their energy reserve.

Surprisingly, very few works consider PCNs in which users send or route payments from light nodes through wireless connections [23–26]. *This thesis aims to fill this gap in the literature by analyzing what challenges arise when we include resource-constrained devices into a PCN.* Specifically, we identify four main challenges to overcome:

- **Challenge #1 (Architecture): Define a PCN architecture considering resource-constrained devices.** We need a network model in which not all nodes can store a copy of the blockchain and verify channel states. Apart from payment channels, this model should consider direct communication between nodes used to request blocks or efficiently update the network graph on demand.

- **Challenge #2 (Channel Security): Adapt channel security mechanisms to the needs of resource-constrained devices.** Attack prevention through constant blockchain monitoring is infeasible for resource-constrained devices that disconnect frequently. We should provide a simple yet effective way to secure payments from such devices while they are offline.

- **Challenge #3 (Simulation): Provide a way to observe payments from resource-constrained devices in unfavorable networking conditions.** Resource-constrained devices often operate under unreliable networking conditions that can lead to interruptions or delays in the payment process when the signal is lost. We should provide a way to simulate these conditions to understand how unfavorable scenarios affect payments.

- **Challenge #4 (Routing): Provide a payment scheme to route payments from resource-constrained devices with low latency.** Resource-constrained devices cannot be assumed to have the same computing power as common nodes. Therefore, finding paths and finalizing payments in these nodes can be too slow for some applications. We need a payment scheme that considers such limitations and speeds up payments from light nodes.

Besides introducing interesting research challenges, developing efficient mechanisms for resource-constrained devices in PCNs would extend the benefits of PCNs to over 13 billion mobile devices that already account for over half of all the traffic on the Internet [27, 28]. In this thesis, we expect to provide meaningful insights

[2]This thesis considers the terms "resource-constrained devices" and "light nodes" interchangeable.

that will help develop new PCN-based applications for light nodes. We highlight that Challenge #4 represents an exciting challenge for readers familiar with classical pathfinding algorithms and flow allocation methods, as payment routing borrows many concepts from transportation networks.

1.2 Contributions

Our work starts with the goal of analyzing the blockchain scalability problem. We compare several state-of-the-art consensus protocols to understand their main advantages and drawbacks, providing a direction on which protocol we should adopt for each use case. We analyze quorum-based protocols, in which a committee of nodes decides what blocks go into the blockchain, and proof-based protocols, in which any node can propose a block if it proves it has the right to do so. This initial work yielded two main publications (see Appendix B for the full list of publications):

- Rebello, G. A. F., Camilo, G. F., Guimarães, L. C. B., Souza, L. A. C., Duarte, O. C. M. B. - "Security and Performance Analysis of Quorum-based Blockchain Consensus Protocols", in 6th Cyber Security in Networking Conference (CSNet'22) - Rio de Janeiro, Brazil, October 2022.
- Rebello, G. A. F., Camilo, G. F., Guimarães, L. C. B., de Souza, L. A. C., Thomaz, G. A., Duarte, O. C. M. B. - "A Security and Performance Analysis of Proof-based Consensus Protocols", in Annals of Telecommunications, no. 7, pp. 517-537, 2021.

The discoveries from this analysis concluded that consensus, regardless of type, is the main bottleneck for transaction processing in blockchains. Hence, we move to consensus-free approaches such as payment channel networks. The first contribution of this thesis is a survey about such approaches, which are formally called *off-chain* or *layer-two* protocols:

Contribution 1: A survey on layer-two protocols for blockchains. We provide a survey on the recent efforts to improve the scalability of blockchains, focusing on layer-two protocols such as payment channel networks and rollups. These technologies process computations off-chain and only use consensus for solving disputes. A large portion of the work addresses the open challenges of payment channel networks, such as payment routing, channel rebalancing, network design strategies, security and privacy, payment scheduling, congestion control, simulators, and support for light nodes.

The survey has been submitted to an international journal and is currently under revision:

- Rebello, G. A. F., Camilo, G. F., Souza, L. A. C., Potop-Butucaru, M., Amorim, M. D., Campista, M. E. M., Costa, L. H. M. K. - "A Survey on Layer-Two Protocols for Blockchains", submitted to IEEE Communications Surveys & Tutorials in April 2023.

As we prepared the survey, we noticed that PCNs have their own challenges, especially regarding resource-constrained devices. The main problem is that the security of channel funds is only guaranteed if both channel parties are online, which cannot be assumed for light nodes. Thus, we propose the following contribution:

Contribution 2: A mechanism to secure channels with light nodes. We formalize the concept of hybrid PCNs, i.e., PCNs with capable nodes and wireless resource-constrained devices, and address the coin theft problem, a vulnerability that affects nodes with intermittent connectivity. We propose a countermeasure based on minimum time windows that lock funds whenever a user disconnects. The duration of the window is proportional to the mean time to recovery (MTTR) of devices, which gives them enough time to reconnect and contest attacks.

This work adapted the default dispute periods of existing PCNs to accommodate light devices [2, 20]. We evaluated our proposal with real channels from Bitcoin's Lightning Network [2] and estimated the MTTR of devices using data from 3G/4G mobile broadband connections [29, 30]. The work resulted in another publication:

- Rebello, G. A. F., Potop-Butucaru, M., Amorim, M. D., Duarte, O. C. M. B. - "Securing Wireless Payment-Channel Networks With Minimum Lock Time Windows", IEEE International Conference on Communications (ICC 2022), Seoul, South Korea, May 2022.

Experimenting with mobile connections showed that its often difficult to predict the behavior of payment channels when they operate under unstable networking conditions. Besides, we missed an automated tool that would accurately simulate payment channel networks with several network topologies and communication protocols. This need generated our second contribution:

Contribution 3: PCNsim. PCNsim is an open-source payment channel network simulator that reproduces the state machine of the Lightning Network on top of the OMNeT++ framework [31]. The simulator allows users to model channel parameters such as channel capacity and routing fees and to test routing protocols on real data obtained from payment datasets. Because OMNeT++ offers a wide range of communication protocols through the INET library[3], PCNSim also supports testing PCNs with different networking protocols in lower layers. PCNsim is available to the scientific community at https://github.com/gfrebello/pcnsim.

We published the details of PCNsim and some simulations of payment routing protocols:

- Rebello, G. A. F., Camilo, G. F., Potop-Butucaru, M., Campista, M. E. M., Amorim, M. D., Costa, L. H. M. K. - "PCNsim: A Flexible and Modular Simulator for Payment Channel Networks", IEEE International Conference on Computer Communications Workshops (INFOCOM WKSHPS), Virtual Conference, May 2022.

Our simulations with PCNsim helped us realize that routing payments efficiently is the most interesting part of dealing with resource-constrained devices in PCNs. Particularly, routing payments from light nodes can be too slow for applications requiring minimum payment latency to work, such as stock markets, cross-chain trades, and electronic toll collection. We contribute to this direction:

Contribution 4: A payment scheme for payments from light nodes. We propose a special payment scheme that reduces confirmation latency when issuing payments from resource-constrained devices. The payment scheme securely offloads payments to gateway nodes which compute routes and forward payments as a service. This saves energy, speeds up payments, and allows light nodes to remain offline most of the time at the expense of a small service fee.

Finally, we also propose new routing algorithms for gateways to route payments considering side constraints:

Contribution 5: GenPulse and MultiPulse. We develop two optimal payment routing algorithms that consider application-specific needs when computing paths. The Generalized Pulse (GenPulse) algorithm finds a constrained shortest path in the graph, i.e., a path that minimizes a routing metric while keeping some side constraints. Multipath Pulse (MultiPulse) extends GenPulse to issue optimal multipath payments via successive flow allocation. GenPulse and MultiPulse can be used in PCNs, for instance, to minimize the routing costs of a payment that is subject to a maximum payment latency or network resource consumption.

Note that despite being proposed for PCNs with light devices, GenPulse and MultiPulse can be used in traditional PCNs. We are currently writing a paper that includes these contributions.

1.3 Thesis Outline

This thesis contains two main parts. In the first part, we present the background needed to understand the challenges of payment channels with resource-constrained

devices. Chapter 2 presents the blockchain scalability problem and our main findings regarding the limitations of consensus protocols. We present the technical concepts of payment channel networks in Chapter 3, including the mechanisms that guarantee the security of payments. The same chapter briefly overviews the main PCN implementations.

In the second part, we answer the research challenges presented in Section 1.1. We address Challenges #1 and #2 in Chapter 4. The chapter formalizes a key vulnerability of PCNs with resource-constrained devices and proposes a countermeasure to avoid fund losses. Chapter 5 addresses Challenge #3 and presents PCNSim, our proposed PCN simulation platform. We provide some demonstrations of the simulator's capabilities. We address Challenge#4 in Chapter 6, which discusses payment routing in PCNs with resource-constrained devices. The chapter presents the rationale behind our proposed payment scheme and the details of the GenPulse and MultiPulse payment routing algorithms. Finally, Chapter 7 concludes the work, presenting future perspectives for resource-constrained devices in PCNs.

Chapter 2

Blockchain Scalability and Layer-Two Protocols

Despite providing disruptive and innovative features, blockchains still present significant latency, power consumption, and transaction throughput issues. The collection of such issues is known as the "blockchain scalability problem" in the literature and is today one of the most important research topics on blockchain technology [15, 18, 32, 33]. In this chapter, we introduce the proper background needed to understand the problem and formalize it through the enunciation of the blockchain scalability trilemma. The chapter elucidates the main reasons why layer-two protocols exist.

2.1 Blockchains and Consensus

A blockchain system is a distributed ledger technology (DLT) that leverages independent validator nodes to record, share, and synchronize transactions in a decentralized peer-to-peer network. Each node in the network stores a local copy of the blockchain, which can verify any transaction since the creation of the system. The blockchain data structure is a chained list of signed transactions batched into blocks. Each block contains a header with the hash of the predecessor block and a content section that stores transactions, as shown in Figure 2.1. A transaction represents an atomic action that transfers assets between a sender and a receiver. To transfer assets, the sender must sign a transaction that transfers the ownership of the assets he/she owns to the receiver and send it to a subset of nodes called *validators*. The transaction is confirmed once it appears in a block, meaning that it has been selected by a validator node and approved by the others. The combination of signed transactions, linked blocks, and replication of the complete data structure provides immutability to any data stored in a blockchain, creating an incorruptible and distributed log of

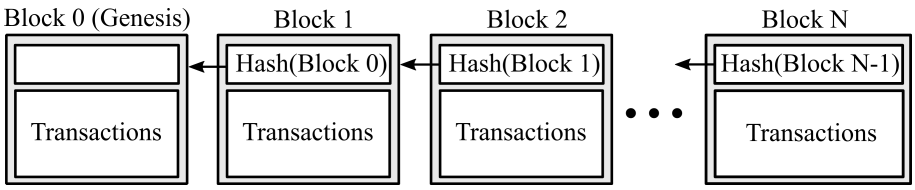

Figure 2.1: Data structure of a blockchain, in which each block is linked to the previous block through a cryptographic hash function. The replication of such structure in independent nodes provides transaction immutability.

transactions.

Consensus in blockchain systems is the process by which the independent validators in the network decide, collectively, whether to accept or refuse the addition of a new block into the blockchain. A blockchain consensus protocol is a distributed algorithm that ensures consensus evolves correctly, adding one new block at a time. Figure 2.2 illustrates how a generic blockchain consensus protocol works. We assume every validator starts at a previously-validated state S. In each round, the consensus leader, i.e., the validator with the right to propose a block, aggregates the received transactions into a block and broadcasts it on the network to be verified locally by the other validators. Upon receiving the proposed block, each validator evaluates it independently and, if approved, adds it to their blockchain, locally reaching the new S' state. When enough validators reach the new state locally, the protocol considers that there has been consensus and that the system as a whole has validated the new block. Hence, S' becomes the current *global* state of the blockchain that all nodes must synchronize with, regardless of their opinion on previous rounds.

Proposing, broadcasting, and verifying the block consume time and energy proportional to the number of validators. A mechanism for defining the consensus leader on each round is also needed. Such procedures and leader-election mechanisms define how fast a block is considered valid and consequently impact the performance of the blockchain. As many works have shown, the core of the blockchain scalability problem lies in the efficiency of consensus protocols, which must ensure the system adds blocks to the ledger in a safe manner [17, 34, 35]. As a reference, the proof-of-work [8] consensus protocol from Bitcoin proposes a block every 10 minutes, yielding 7 transactions per second. This is due to the costly mathematical challenge of deciding who has the right to propose a block.

2.2 The Blockchain Scalability Trilemma

The blockchain scalability problem is a generic umbrella term that encompasses the challenges of improving blockchain performance. For the purposes of this paper, it helps to narrow down such concept to a more specific definition based on the *blockchain*

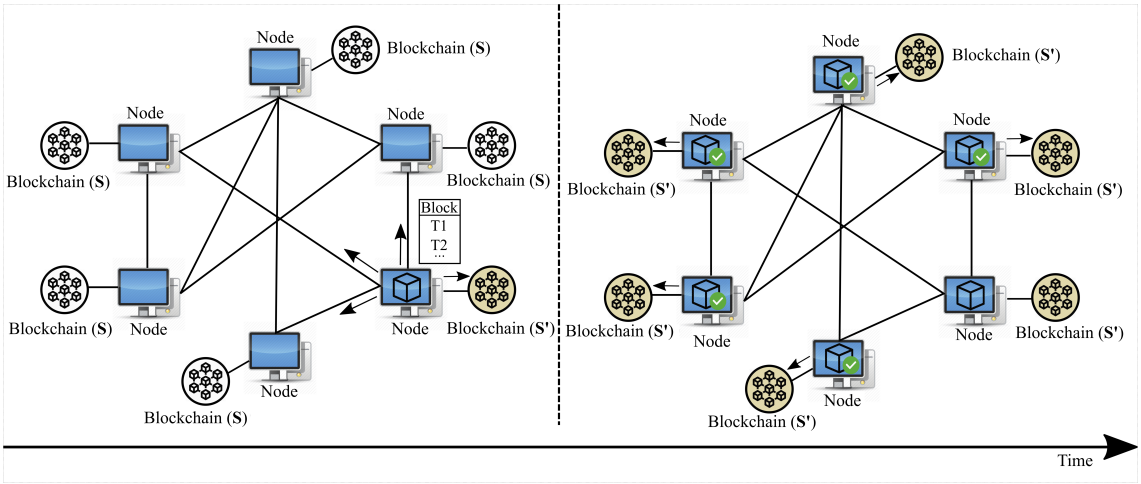

Figure 2.2: The validation of a block using a generic consensus protocol. On each round, the consensus leader proposes a new block that changes its local state from S to S' and broadcasts it to the network. The other participants independently verify and add the proposed block to the blockchain, replicating the state S' consistently.

scalability trilemma[1], a term coined by Ethereum's founder Vitalik Buterin [12]:

Definition 1. *(The blockchain scalability trilemma) [12].* Given the following properties of blockchains:

- **Scalability**: the capacity to process transactions at high throughput,
- **Decentralization**: the capacity to process transactions without relying on trusted parties or small groups,
- **Security**: the capacity to successfully resist collusion attacks,

the blockchain scalability trilemma is a conjecture that states that no blockchain system can provide all properties simultaneously. Consequently, every blockchain forfeits at least one property at any given time.

We highlight that the trilemma refers to Vitalik Buterin's notion of scalability instead of the classical concept of scalability in distributed computing. The latter concept, which in blockchains corresponds to the ability of the consensus protocol to maintain throughput even when the number of validators significantly increases, is captured by the decentralization property. Henceforth, we adopt the term "scalability" to refer to Buterin's concept and "decentralization" to refer to the concept of distributed computing.

We illustrate the trilemma in Figure 2.3. The rationale behind it stems from the observation that the validation of transactions in a blockchain system occurs, by definition, through the agreement between the validators of the system. On the one hand, the more nodes participate in consensus decisions, i.e., the more decentralized

[1]Some authors refer to the blockchain scalability trilemma as simply "the blockchain trilemma" or "the scalability trilemma" [13–15, 36].

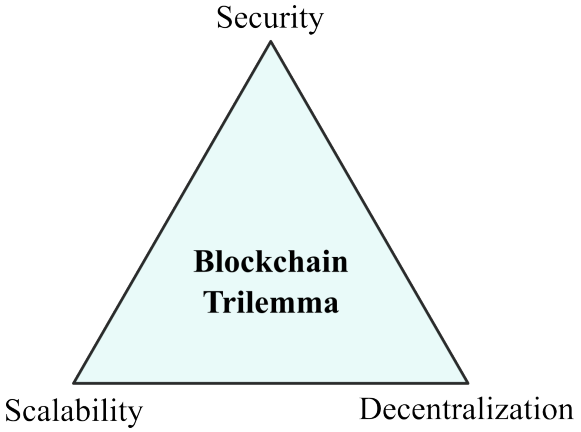

Figure 2.3: Illustration of the trilemma observed in blockchain-based systems. The trilemma states that no consensus protocol can simultaneously provide security, scalability (measured in transaction throughput), and decentralization (measured in the number of nodes participating in consensus). Graphically, all blockchains correspond to a point inside the trilemma triangle.

the system becomes, the more complex and time-consuming the decision-making and broadcasting of messages in the network. On the other hand, reducing the number of validators to improve throughput concentrates the decision power on fewer agents, reducing the level of decentralization and increasing the financial monopoly of the network. Some protocols try to provide high throughput with many validators by allowing multiple parallel blocks to be approved at the same time [37, 38]. This event is known as a *fork* in the blockchain and compromises security since conflicting transactions in different branches could be considered valid. Most protocols solve forks by finalizing a block and discarding the others, but this process also takes time. Besides, transactions in discarded blocks are rolled back and become untraceable, meaning that the security of a transaction is only guaranteed when its block is finalized. Despite being a conjecture based on empirical observations, the trilemma consistently occurs in all significant known blockchain systems [1, 39, 40].

2.3 The Trade-off of Consensus Protocols

As security is essential in blockchains, in practice, the trilemma becomes a dilemma for consensus protocols: the protocol needs to choose between scalability, measured in the number of transactions processed per second, and decentralization, measured in the number of consensus validators. This choice is the primary separator of consensus protocol types, which can be categorized into proof-based protocols, committee-based (or quorum-based) protocols, and hybrid protocols.

Proof-based protocols adopt decentralized mechanisms to define who has the right to propose a block, allowing any user to participate in the process. However, these

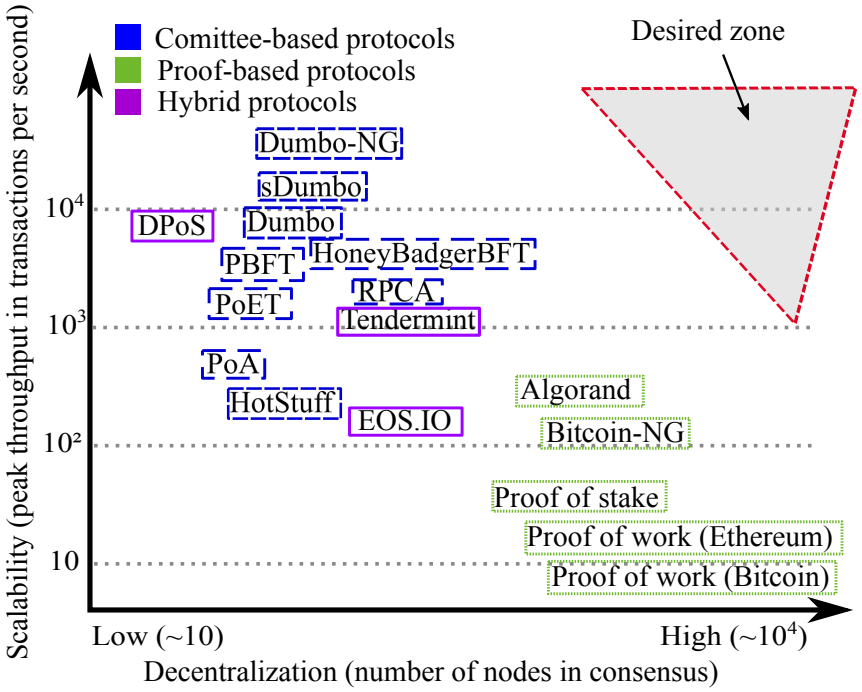

Figure 2.4: Comparison between the main consensus protocols of blockchain-based systems. The observed trade-off between performance and decentralization makes it difficult to propose a scalable protocol that is tolerant of collusion attacks.

protocols achieve low transaction throughput because they need to introduce spam control tools to mitigate forks in the blockchain and solve forks that occur. Thus, proof-based protocols are challenging to scale but very decentralized, making them more suited to public systems with many users [39, 41, 42]. The main systems that use this type of consensus are cryptocurrencies, such as Bitcoin [8], Ethereum [9], or IOTA, [37].

Conversely, committee-based protocols elect a group of special validators that propose blocks through direct communication [43–50]. The choice of who participates in the committee can be made in several ways, such as random selection or an election based on the number of coins each user invested. The selection of a committee sacrifices decentralization, as only some users participate in the decision, but increases the number of transactions processed per second since decisions are independent of time-consuming computational mechanisms and spam control. Committee-based protocols work better in systems that already expect some level of centralization, such as consortia of companies, banks, and governments. The prominent representatives of this type of system are Hyperledger Fabric [51], Hyperledger Sawtooth [52], and Ripple (RPCA) [53]. The design of committee-based protocols must include security mechanisms to avoid collusion and denial-of-service attacks.

Several consensus protocols try to solve scalability through hybrid solutions combining the best proof-based and committee-based consensus approaches [54–57]. The main objective of such protocols is to provide each property at a specific phase of

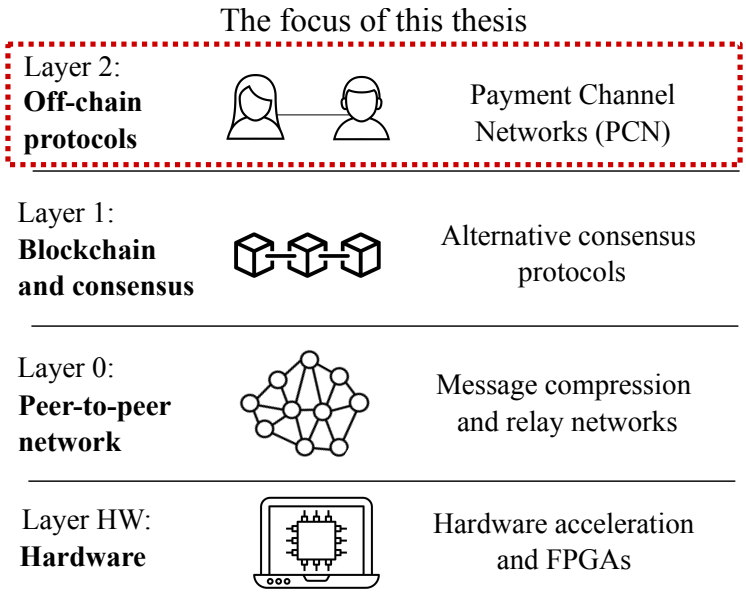

Figure 2.5: Layered stack of blockchain systems, adapted from Gudgeon et al. [1]. The right side of the figure names a few scalability solutions that focus on the corresponding layer. Off-chain protocols operate on top of the blockchain and consensus layer.

consensus, leveraging extra-consensus mechanisms to mitigate possible vulnerabilities. However, the hybrid approach suffers from the same trade-off between scalability and decentralization of other approaches, reaching intermediate levels in both aspects for its main representatives, EOS.IO [54] and Tendermint [55].

We can represent the scalability-decentralization trade-off graphically by comparing the main blockchain consensus protocols shown in Figure 2.4. Despite the efforts on blockchain research over the years, no consensus protocol consistently provides a throughput of thousands of transactions per second with high decentralization [39, 40, 58]. With consensus-based solutions so far, the desired "ideal zone" that would allow scaling blockchain systems without compromising their decentralization seems unachievable.

2.4 Off-chain Protocols

The limitations of consensus-based approaches induced an orthogonal research area for improving blockchain scalability, which consists of so-called *off-chain* protocols [13–15, 59, 60]. Off-chain protocols are also named *layer-two* protocols as they are primarily built on top of the blockchain stack depicted in Figure 2.5. Their prominent representatives are payment channel networks, which we present in Chapter 3.

Although proposals for lower layers are out of the scope of this thesis, such proposals receive less attention from the community because they involve modifications that are difficult to implement. For instance, improvements in the hardware

layer imply acquiring equipment that is often unaffordable for common users [61–63]. Changes to the peer-to-peer communication protocols can cause soft forks[2] or even hard forks[3] in the blockchain, which take time to be accepted by the cryptocurrency community [64, 65]. Improvements in layer one are even more radical as they involve building new consensus protocols or altering the blockchain structure itself, creating whole new systems [37, 54, 56].

Layer-two scalability solutions, on the other hand, do not modify blockchain systems at all. They leverage blockchain core functionalities, such as scripts and smart contracts, to implement security mechanisms that validate off-chain transactions. Such validation mechanisms are invisible to the blockchain, which sees transactions from a layer-two service as ordinary transactions. Besides, off-chain protocols only publish transactions when needed, avoiding the cost and latency of consensus most of the time. This simple but powerful characteristic made layer-two proposals emerge in recent years as a solution with great potential to break the scalability-decentralization trade-off without causing significant impacts to end users [13–15, 59, 60].

2.4.1 Defining Layer Two

Before proceeding into payment channel networks, we highlight that the definition of layer two is still under discussion. For some authors, any mechanism that avoids publishing transactions in the main blockchain is layer two, as such solutions neither rely on the primary consensus protocol to process transactions nor make all transaction data publicly available [13–18]. This perspective categorizes proposals that validate transactions through faster secondary blockchains, such as sidechains, as layer-two protocols. Conversely, other authors define layer-two protocols as protocols implementing their own off-chain consensus-free validation rules [1, 36, 59, 66]. Under this definition, layer-two protocols only rely on consensus to settle disputes that could not be solved via off-chain validation.

We adopt the latter concept as it enhances the innovation of off-chain validation. Besides, associating consensus with layer one provides a more precise separation between layers. In particular, we adopt the definition of layer-two protocols as proposed by Gudgeon et al. [1]:

[2]Soft forks are forward-compatible modifications that do not compromise the functionality of old software versions.

[3]Hard forks are backward-incompatible modifications to the blockchain which force all nodes to update their software.

Definition 2. *(Layer-two protocols).* A layer-two protocol is a protocol that allows transactions between users through the exchange of authenticated messages via a medium that is outside of but linked to a layer-one blockchain. Authenticated assertions are submitted to the main chain only in cases of a dispute, with the main chain deciding the outcome of the dispute. Security and non-custodial properties of a layer-two protocol rely on the consensus protocol of the main chain.

Hence, we consider proposals that modify consensus or use consensus as the default validation mechanism as layer one, including sidechains and cross-chain protocols. To the best of our knowledge, *payment channel networks* [2, 20], *optimistic rollups* [67], and *zero-knowledge rollups* [68, 69] are the only currently known layer-two protocols for blockchain scalability. In the next section, we dedicate much of this work to presenting payment channel networks, the most popular layer-two protocol for exchanging assets efficiently in cryptocurrencies. The payment channel network use case covers most of the characteristics of layer-two protocols, so the reader should understand how the main off-chain functionalities work even though rollups are out of the scope of the thesis.

2.5 Summary

The blockchain scalability problem is the main reason blockchains struggle to substitute centralized payment methods like credit cards. In this chapter, we showed that the leading cause of the problem is the scalability-decentralization trade-off of consensus protocols, which can only provide high throughput with centralizing decisions. Consequently, no consensus protocol provides the necessary levels of scalability and decentralization to address the needs of payments globally. Layer-two protocols are a promising solution to this problem, as they implement off-chain validation procedures that offload consensus and enable fast payments at a low cost.

Chapter 3

Payment Channel Networks

Payment channel networks are a solution to the blockchain scalability problem presented in the previous chapter. This technology operates in layer two, establishing off-chain communication channels where users can freely send payments without validating them through a consensus protocol. Such characteristic reduces the latency of payments, which now depends mainly on communication latency between users. Payment channels are a particular case of state channels in which the traded assets are coins [70].

3.1 Overview of Payment Channels

The main idea of payment channels is only to publish transactions in the blockchain when needed. The blockchain becomes a security service that provides a secure starting point for the payment channel and possibly settles disputes involving users.

Figure 3.1 shows a payment channel's default three-phase life cycle. In the channel creation phase, two users, Alice and Bob, issue a funding transaction, agreeing to transfer some of their coins to a joint address they cooperatively control. These coins can be used to issue transactions inside the channel but become unavailable for other transfers in the blockchain for the entire existence of the channel. Bob and Alice also agree on a timelock window W that either user must wait in case he/she unilaterally closes the channel (we explain why we need a timelock window in Section 3.2.3). Then, when the channel is open, the users continuously update their balances in the channel via private commitment transactions. Commitment transactions are not published in the blockchain and, consequently, produce low-latency payments. At any time, either user can decide to trigger the channel closing phase by publishing the latest commitment transaction, which contains the latest channel state, into the blockchain. Once validated, this transaction creates an Unspent Transaction Output (UTXO) in the blockchain that transfers the coins to their respective parties on-chain. Thus, the blockchain only sees the channel-opening and channel-closing

transactions and regards them as standard transactions. Consequently, these are the only transactions that go through consensus validation and are subject to high latency and transaction fees.

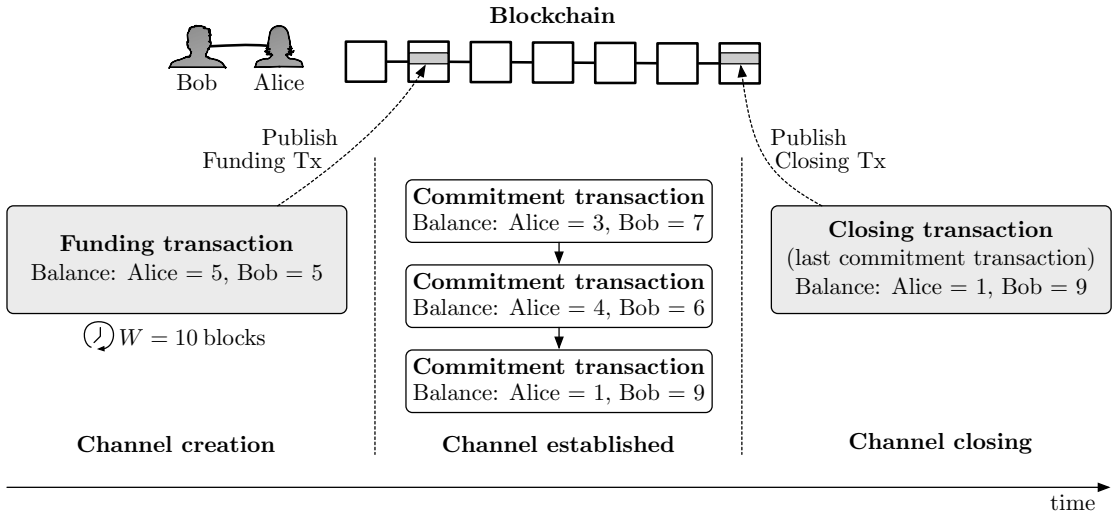

Figure 3.1: Payment channel operation. Users Alice and Bob contribute 5 coins each to issue the funding transaction and create a payment channel. After the channel is open, Alice and Bob can exchange coins by issuing private commitment transactions that rewrite their balances in the channel. For example, Alice sends 2 coins to Bob by signing the first commitment transaction, which changes her balance to 3. Alice or Bob can publish the commitment transaction containing the most up-to-date balances to close the channel and claim the coins on-chain.

Payment channels, like blockchains, presume no mutual trust between parties. Therefore, there must be a secure mechanism to ensure coins remain safely in possession of their respective owners. The system accomplished coin security with *multi-signature* (or *multisig*) payment schemes [71]. In a multi-signature scheme, multiple users need to sign a transaction for it to be valid; otherwise, the coins remain locked. A multi-signature scheme in which at least n out of m pre-defined users need to sign the transaction is called an *n-of-m multisig policy*.

When establishing a channel, the funding transaction contains a 2-of-2 multi-signature payment policy, i.e., each new transaction that wishes to spend its coins must contain the signature of both users involved in the channel. To issue a commitment transaction within the channel, Alice creates a transaction that spends coins from the funding transaction, signs it, and transfers the signed transaction to Bob. The signed transaction guarantees to Bob that Alice agrees with the current channel balances. Bob can then sign and publish the transaction in the blockchain to redeem his funds. Bob or Alice cannot steal coins from the channel with this scheme, as any published transaction would need the other party's consent.

However, locking coins in a transaction requiring both participants' signatures introduces a vulnerability: if Bob refuses to sign transactions or disconnects, he can

lock Alice's coins forever. Payment channels avoid this by forcing Alice and Bob to generate a transaction that spends the funding transaction with the initial balances. Users create and exchange this refund transaction before the funding transaction is published. Thus, Bob has a refund transaction signed by Alice, and Alice has a refund transaction signed by Bob before the channel is open. In case one of the parties becomes unresponsive, the other party can issue the refund transaction on the blockchain and reclaim their rightful coins.

3.2 Channel State Updates

The refund transaction serves as an initial guarantee but must be replaced with other transactions that reallocate balances in the payment channel. For this substitution to occur safely, the system must correctly revoke the latest transaction and introduce a new one, updating the channel state. Therefore, an essential aspect of payment channel security is channel state updates [1, 2].

If channel updates are incorrectly performed, a malicious user can publish an old state that benefits him. For example, users perform three off-chain commitment transactions in Figure 3.1. First, Alice issues a transaction, TX_1, sending 2 coins to Bob. Then, Bob issues the second transaction, TX_2, sending 1 coin back to Alice. If Bob is an attacker, at this point, he can try to close the channel with the first transaction, which guarantees him a balance of 7 coins instead of the 6 coins he has in the current state. Finally, Alice issues the last transaction, TX_3, sending 3 coins to Bob. Now, Alice could send the refund transaction (not shown in the figure) to the blockchain, claiming the 5 coins she initially had instead of the 1 coin she currently has.

This vulnerability happens because the blockchain does not keep track of off-chain transactions. Therefore, it must be made aware of which transaction contains the most recent state. Payment channels, however, do not need to maintain a complete ordering of off-chain transactions like blockchains; it suffices to keep track of the latest state. In the example, it is enough to enforce that only TX_3 is valid, i.e., there must be a safe way to revoke TX_1 and TX_2. We explain below several proposals to solve the problem of state updates in payment channels [2, 72, 73].

3.2.1 One-way Payment Channels (Spillman Channels)

In 2013, Spillman [72, 74] proposed one of the first state update mechanisms for BitcoinJ[1] payment channels, which only work for one-way payments. This mechanism

[1]BitcoinJ is a Java implementation of the Bitcoin protocol. Available at https://bitcoinj.org/

imposes conditions to claim coins using the programming language of Bitcoin, the Bitcoin script [75].

In Spillman channels, users A and B generate a refund transaction that contains the following conditions to close the channel: (i) the locked coins can be redeemed after time t counted from the publication of the funding transaction, or (ii) the coins can be redeemed immediately if the two parties agree with the refund. The first condition guarantees that coins cannot be locked in the channel forever. In contrast, the second condition guarantees that the two users will receive a refund immediately if both sign a transaction, proving they cooperated. After the channel is open, user A starts to send coins by issuing commitment transactions to user B, who can sign and publish the transactions in the blockchain if needed.

This channel design works only as a one-way channel. While user B receives transactions signed by A, the same does not happen in the opposite direction. Furthermore, even if user B returns a coin to user A, there is no guarantee that B will not publish an old transaction on the blockchain in which he had a higher balance. The correction of state replacement is unilateral and incentive-based since the user who receives the coins can only lose funds by publishing an old transaction. Any rational user who receives payments will always post the latest state as it benefits them the most [1].

The creation of a payment channel that works only in one direction, however, restricts the potential applications of this technology. Most day-to-day applications require payments in both directions, e.g., refunds for purchases or cashback applications. Furthermore, two users who share a payment channel can take on independent roles, buying and selling products to each other. In this case, one-way channels eliminate the possibility of safely sending payments in opposite directions. Users who want to reverse the roles of sender and recipient would have to create one channel in each way, which implies more locked coins, more transaction fees, and more time to establish the channels.

3.2.2 Bidirectional Payment Channels (1/2): Limited Duration Model

One of the early ideas to secure bidirectional channels was to create time lock policies that discourage users from issuing old transactions [73]. In this model, all off-chain transactions have a time lock, i.e., if published, the coins in the transaction can only be spent after a time window W counted from its publication in the blockchain. The model adopts the blockchain as a reference to guarantee synchronization, with the time window defined in the number of elapsed blocks. The time window decrements each time the payment direction reverses. Thus, in a channel between two users

A and B, user A can issue a transaction TX_1 to B with a 30-minute time lock, approximately 3 blocks on the Bitcoin blockchain (recall that one block takes on average 10 minutes, see Section 2.1). If B wants to pay A, B issues TX_2 with a time lock of 20 minutes, approximately 2 blocks in the Bitcoin network. That way, if B publishes the previous transaction, TX_1, in the blockchain, user A can issue TX_2 and redeem the coins before B since TX_2 has a shorter time lock.

Despite safely ensuring state replacement, this model has two disadvantages: (i) user A must be online and constantly checking the blockchain to monitor the actions of B, and (ii) the size of W defines the duration of the channel. The definition of W, done by the two users before they open the channel, represents a trade-off. A high value for W allows for more state updates but locks coins for a longer time when a user closes the payment channel. A low value of W allows few channel updates, reducing the channel expiration date. Besides, users must define W without knowing how many payments will traverse the channel.

3.2.3 Bidirectional Payment Channels (2/2): Standard Model

Poon and Dryja [2] proposed a state substitution model that has become the standard for payment channels. The model creates bidirectional channels while discouraging malicious behavior through financial punishment. If it is proven that a user published an old state in the blockchain, the channel counterpart can contest the state and redeem all the coins in the channel, including the attackers.

In Poon and Dryja's model, each direct payment generates a pair of asymmetric commitment transactions: user A obtains a commitment transaction signed by user B, and user B obtains a commitment transaction signed by user A. The transactions have an associated secret and the following condition: if the counterpart reveals this transaction's secret within a period of W blocks counted from the transaction publication, all the coins in the channel go to him/her. To issue new payments on the channel, users change the state by exchanging signatures and revealing the secret associated with the previous transaction. Thus, if any user publishes an old state, the other user can immediately claim all the coins using the previously-revealed secret. The user who closed the channel can only redeem coins after W blocks. In practice, the transaction secret is a private key that can transfer the commitment transaction outputs to the address of the user who contested the transaction. The key verification is done automatically using Bitcoin script as a root of trust, public to both participants.

The procedure for replacing states in this model executes as follows. Assume a channel between Alice and Bob in which, initially, each user has 5 coins, and Alice wants to send 1 coin to Bob. To send the payment, Alice generates an asymmetric

key pair $(PK_A(s), SK_A(s))$, where $PK_A(s)$ represents Alice's public key and $SK_A(s)$ represents Alice's secret key for the channel state s. Bob performs the same procedure, generating a key pair $(PK_B(s), SK_B(s))$. The two parties exchange their public keys. Then, Alice creates a commitment transaction containing the following conditions:

(i) If this transaction is published, Alice can claim 4 coins immediately;

(ii) If this transaction is published, Bob can claim 6 coins after a period W or immediately if Alice authorized it;

(iii) If this transaction is published, Alice can claim 10 coins if she proves she knows Bob's secret $SK_B(s)$ in time $t_i < W$. Alice hard-codes $PK_B(s)$ into the transaction so she can prove knowledge of $SK_B(s)$ through asymmetric encryption.

Next, Alice signs the transaction and sends it to Bob with her secret key of the previous state, $SK_A(s-1)$. Bob concurrently performs the same procedure as Alice, generating the transaction with the following conditions:

(i) If this transaction is published, Bob can claim 6 coins immediately;

(ii) If this transaction is published, Bob can claim 4 coins after a period W or immediately if Bob authorized it;

(iii) If this transaction is published, Bob can claim 10 coins if she proves she knows Alice's secret $SK_A(s)$ in time $t_i < W$. Bob hard-codes $PK_A(s)$ into the transaction so he can prove knowledge of $SK_A(s)$ through asymmetric encryption.

Bob signs the transaction, sends it to Alice, and reveals his secret key for the previous state, $SK_B(s-1)$. This secret-revealing protocol ensures that only conditions (i) and (ii) can be triggered if both parties act honestly since $SK_A(s)$ and $SK_B(s)$ are shared in the next state update. However, if either Alice or Bob publishes an old state, the other party can immediately trigger condition (iii) with the previous secret keys $\{SK_A(1), ..., SK_A(s-1)\}$ and $\{SK_B(1), ..., SK_B(s-1)\}$ that have already been revealed. Because they allow canceling old transactions through financial punishment, the key pairs generated per state are called *revocation keys*. Transaction outputs that can be revoked with revocation keys are called revocable deliveries.

Payment channels create revocable deliveries through Revocable Sequence Maturity Contracts (RSMC) [2]. RSMCs are simple if-then-else conditions that create two possible paths for a revocable transaction. We illustrate with an RSMC example written in Bitcoin script:

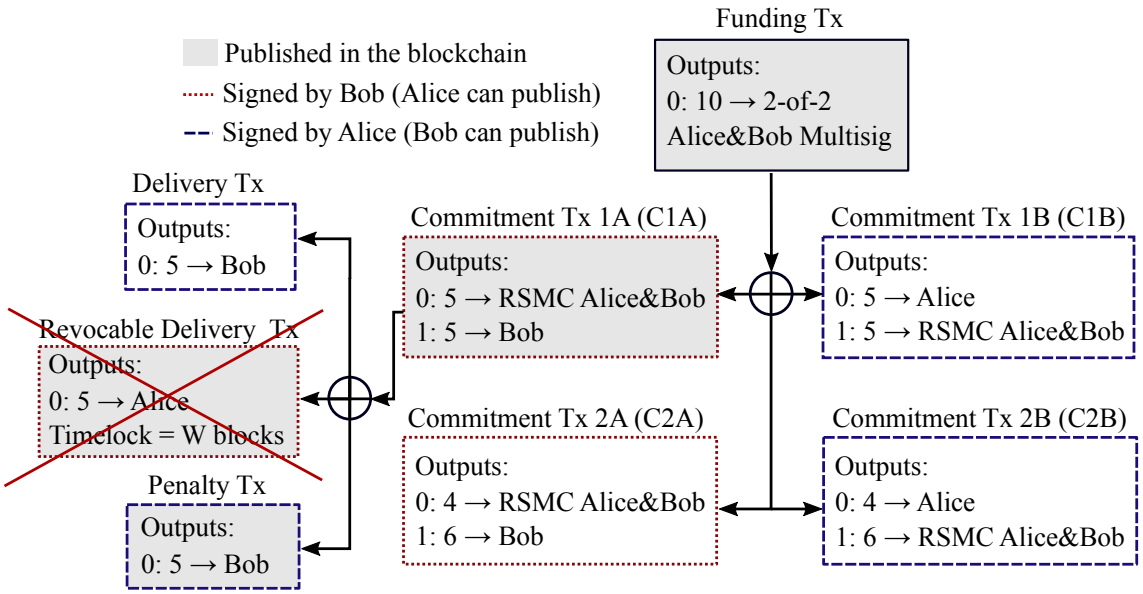

Figure 3.2: An example of state revocation in payment channels. Alice closes the channel with C1A, in which she had a higher balance than the current state, and then attempts to withdraw the coins with a revocable delivery transaction. Bob detects the attack and issues the penalty transaction within W blocks as a response, revoking Alice's delivery and transferring her coins to his address. He can still claim his rightful coins with a common delivery transaction.

```
OP_IF
    # Penalty transaction path
    <RevocationPubKey> # Push revocation pubkey into the stack
OP_ELSE
    # Revocable delivery transaction path
    W # Push W into the stack
    OP_CHECKSEQUENCEVERIFY # Stop if the current block < W
    OP_DROP # Remove W from the stack
    <LocalPubKey> # Push the local public key into the stack
OP_ENDIF
OP_CHECKSIG # Compare signature with the key in the stack
```

In this example, any user that provides the transaction's revocation key can claim the payment immediately by triggering the penalty transaction path. Else, if enough time passes, the user who published the transaction can claim the coins through the revocable delivery path. The RSMC protocol imposes no limits on the channel's lifetime. However, it guarantees that any party can close the channel unilaterally if needed, with the drawback of waiting for a dispute period of W blocks. The time window W that sets the duration of the dispute period is called `to_self_delay` in the Lightning Network [2, 76].

Finally, we illustrate an example where Alice publishes a revoked state and gets

punished by Bob in Figure 3.2. Note that the protocol implies Alice and Bob must store all revocation keys since the channel's opening and generate a new key pair per transaction. Alice and Bob must also be online and read the blockchain to verify if the other party tried to cheat. Requiring constant monitoring can be an issue for use cases where Alice or Bob need to disconnect, as discussed in Chapter 4.

3.3 Payment Channel Networks

Despite providing fast and secure payments, a payment channel works for a pair of users. This solution alone does not solve the blockchain scalability problem as it would require users to establish channels to every payment destination [2]. For users who need to transact with multiple destinations, this means: (i) having many coins locked in the blockchain to allocate funds across multiple channels and (ii) paying a transaction fee and waiting for the consensus delay for each funding transaction. Such requirements hinder using payment channels alone for cases where payments must reach multiple entities.

The solution which actually helps to improve scalability in blockchains is a *payment channel network* (PCN):

Definition 3. *(Payment Channel Network).* A payment channel network is the interconnection of the payment channels created by users. PCNs enable multi-hop payments that traverse payment channels, eliminating the need to create a channel per destination. Multi-hop payments allow users to transact with each other quickly with low fees, even if they do not share a direct channel.

Figure 3.3 shows an example of a PCN with 5 participants. In the picture, Alice wants to transfer 5 coins to Charlie, with whom she does not have a payment channel. Alice, however, has a channel with Bob, who has a channel with Charlie. Hence, Alice can transfer 5 coins in her channel with Bob, who then transfers 5 coins in his channel with Charlie, completing Alice's payment. Note that payment channels are independent, meaning that Bob cannot transfer the coins from his channel with Alice to the channel with Charlie. He receives Alice's commitment transaction on his channel with Alice and generates a commitment transaction of equal value for Charlie. Note that this implies Bob has at least 5 coins of balance in his channel with Charlie, or he cannot forward Alice's payment.

3.3.1 Securing Payments with Hashed Timelock Contracts

An obvious concern when sending payments through untrusted intermediaries is how to guarantee the payment will reach its destination. For example, in Figure 3.3, Bob

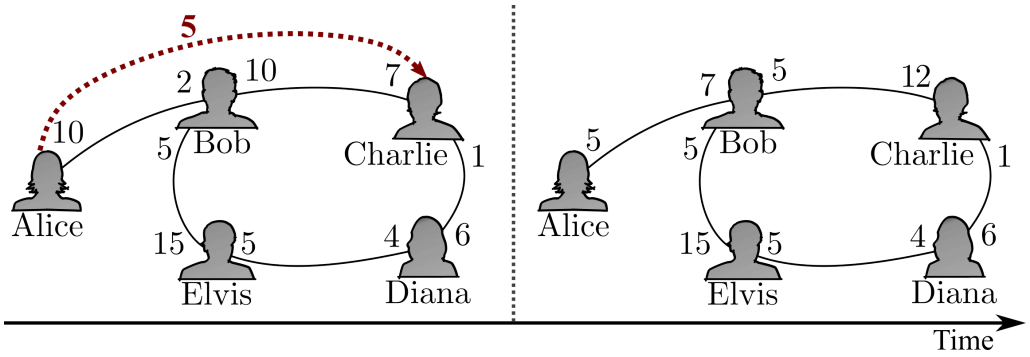

Figure 3.3: Example of payment in a PCN. To send Charlie 5 coins, Alice uses the existing channel with Bob, who forwards the coins to the destination. The payment modifies the balance of the channels involved in the payment path.

could easily steal Alice's coins if he decides not to forward the coins on his channel with Charlie. Since PCNs assume no nodes in the network can be trusted, there must be a secure and scalable payment scheme that prevents intermediaries from stealing payments.

PCNs prevent payment stealing with a special type of contract called a *Hashed Timelock Contract (HTLC)*. An HTLC is a coin-locking mechanism that enables conditional payments in a channel. HTLCs are said to be locked by *hash* and *time* because they impose the following conditions to a payment [2, 77]:

1. **Secret disclosure.** The payment has an associated arbitrary hash value. If Bob reveals the secret[2] that generated this hash, he can claim the payment;

2. **Timeout.** If a long time passes, the previous condition is considered null. Alice can assume Bob gave up the payment and claim the coins back.

Thus, HTLCs guarantee that each payment can only be claimed by revealing the payment preimage or releasing the locked coins after a timeout. This mechanism establishes trust among users involved in a multi-hop payment. For instance, assume Alice wants to send a conditional payment to Charlie using Bob as an intermediary in Figure 3.4. First, Charlie generates a random value x, calculates its hash, $y = H(x)$, using some known hash function, and sends y to Alice. Alice then establishes an HTLC of 1.1 BTC (Bitcoins) with Bob, informs Bob that Charlie is the next hop, and promises to deliver the money to Bob if Bob reveals x within 9 blocks (approximately 90 minutes in Bitcoin). Bob then generates an HTLC with the same conditions but a slightly lower value (1.0 BTC) to Charlie, setting its timeout to 8 blocks (approximately 80 minutes in Bitcoin). Charlie, who generated the x value, reveals x within 8 blocks to Bob and claims his payment. Bob, in turn, reveals x within 9

[2]The payment secret is also called a payment preimage [2, 76]. We henceforth adopt this name to avoid confusion with transaction secret keys.

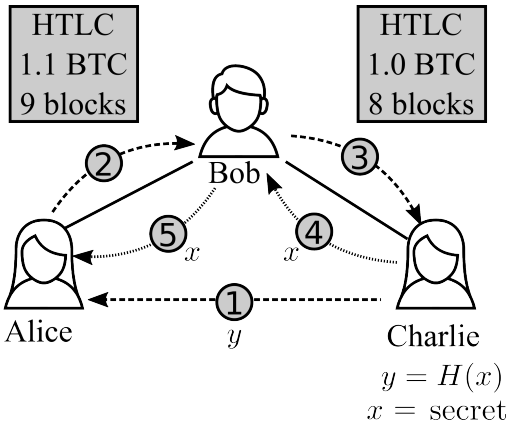

Figure 3.4: Steps involved in a multi-hop payment in an example PCN. 1) Charlie, the recipient, generates a preimage and sends the hash of this preimage to Alice, the sender. 2) Alice establishes a Hashed Timelock Contract (HTLC) with Bob promising she will deliver 1.1 BTC (Bitcoin) if he reveals the preimage x within 9 blocks time. 3) Bob performs the same procedure with Charlie, slightly reducing the value and time limit. 4) Charlie reveals the preimage to Bob and claims the money Bob promised. 5) Bob reveals Charlie's preimage to Alice, claiming the money she promised and finishing the payment.

blocks to Alice to claim his payment in the channel with Alice. Bob's claim finalizes the payment, guaranteeing all channel balances were correctly updated.

Note that Bob promised a slightly lower value to Charlie than Alice promised him. This difference in HTLC values is a *routing fee*[3] charged by Bob to forward Alice's payment in his channel with Charlie. Routing fees serve as a financial incentive to forward payments correctly. They are always equal to the difference between the incoming HTLC's and outgoing HTLC's values (respectively, the HTLC from Alice to Bob and the HTLC from Bob to Charlie in the figure).

Nodes announce their routing fees per channel through gossip messages in the network. Senders pay fees by setting the value of the first HTLC as

$$HTLC_V = P_V + \sum_{(i,j)\in\pi} \phi_{ij}, \tag{3.1}$$

where P_V is the payment's value and ϕ_{ij} are the routing fees charged for forwarding the payment in each channel (i, j) of the payment path $\pi = ((s, i_1), (i_1, i_2), ..., (i_n, t)) \in \mathbb{R}^n$. Then, the following HTLCs subtract the routing fees per hop. Routing fees ensure that every intermediary that acts honestly will receive more than they have forwarded.

In addition, each intermediary in the payment path must set a timeout shorter than the timeout of the previous hop. This difference ensures that all intermediaries

[3]Routing or payment fees are off-chain fees charged by intermediaries to forward payments. They should not be confused with the transaction fees needed to publish a transaction in the blockchain.

will have enough time to redeem coins. Thus, if Alice generates an HTLC with timeout T with Bob, Bob must generate an HTLC with timeout $T' < T$ with Charlie; otherwise, Bob risks losing funds. Similar to routing fees, the payment sender must set the timeout of the first HTLC as

$$HTLC_T = P_T + \sum_{(i,j)\in\pi} \Delta_{ij}, \tag{3.2}$$

where P_T is the desired payment timeout and Δ_{ij} is the minimum difference between the timeouts of any incoming HTLC and an outgoing HTLC in channel (i, j). This minimum timeout difference ensures there is enough time for the intermediary to claim his/her coins when he/she receives the payment preimage in channel (i, j). Nodes announce the time differences per channel as with routing fees. The timeout of the payment, P_T, and the per-channel timeout differences, Δ_{ij}, are respectively called `cltv_expiry` and `cltv_expiry_delta` in the Lightning Network [3, 76]. The names come from the `CheckLockTimeVerify` operation in Bitcoin script, which verifies HTLC timeouts [78].

We highlight that locking payments through HTLCs differs from locking funds in funding transactions; instead of locking coins in the blockchain, HTLCs lock part of the coins already in the channel, reducing the channel's available balance. Consequently, the number of coins the node previously allocated in the payment channel bounds the maximum amount of coins a node can forward in an HTLC. In the example of Figure 3.3, Bob could not forward payments of more than 10 coins in the channel with Charlie.

Despite being called contracts, the mechanism of HTLCs follows a simple logic that can be implemented even in systems that do not support smart contracts. This easiness of implementation increases the applicability of payment channel networks, which work in any blockchain system. For example, HTLCs in Bitcoin are simple if-then-else clauses inside the script of commitment transactions [2]. HTLCs are the core enabler of multi-hop payments, which, in turn, make PCNs scalable.

3.3.2 Payments vs. Transactions

Now that we have presented the concept of HTLCs, it helps to define the concept of an off-chain payment precisely:

Definition 4. *(Off-chain payment).* An off-chain payment, or simply a payment, is the off-chain transfer of assets between a sender and a recipient through the channels of a payment channel network. If the payment is multi-hop, it establishes a sequence of HTLCs, called a payment chain, in the channels of the payment path. The respective channel parties can claim each HTLC on-chain.

Such a payment definition is needed to clarify the difference between payments and transactions, which we omitted in the previous sections for simplicity. On the one hand, a transaction is the signed data exchanged between two parties that can be published to transfer assets on-chain, e.g., the commitment transactions in a channel. This definition is equivalent to a *direct* off-chain payment. On the other hand, a *multi-hop* payment is an off-chain transfer that needs an end-to-end sequence of HTLCs to occur. Because HTLCs reside inside commitment transactions on each channel, we say that a multi-hop payment involves one transaction per channel in the payment path. Henceforth, we refer to this definition whenever we mention the terms "payment" or "off-chain payment".

3.4 PCNs: The Lightning Network

The Lightning Network, proposed by Poon and Dryja [2] for Bitcoin in 2016, is the first large-scale implementation of a PCN. The system issued its first channel-opening transaction in 2017 and since then has become the most significant known PCN, with more than 16,000 nodes and 80,000 payment channels distributed around the world [15, 66, 79]. Besides, the recent topological analysis illustrated in Figure 3.5 indicates the network is still rapidly growing, with the number of nodes and channels tripling from January 2020 to August 2021 [80]. The graph is built with data from gossip messages exchanged inside Lightning [81]. Due to its size and importance, the Lightning Network is the reference model of a PCN today, providing real-time digital payments for several applications [82]. The development of the Lightning Network is even a critical factor for adopting Bitcoin as an official payment method in El Salvador [19].

Lightning has a strong focus on providing user anonymity, just like Bitcoin. As such, it implements several privacy-preserving mechanisms, such as user identification through public keys only. Payments use onion routing [83], a protocol that encrypts packets once per hop and is mainly known for its privacy guarantees in the Tor network implementation [84]. Onion routing ensures that payment intermediaries only know the next hop in the payment path.

The network implements bidirectional payment channels precisely as described in Section 3.2.3 [2]. However, users can publicly announce their channels or keep them private to ensure privacy. The disclosure of channels in the network occurs through channel announcement messages that the network broadcasts in a gossip fashion [3]. The nodes in Lightning build a graph from the received messages containing the active channels with their respective participants, fees, and capacities. Private channels are not advertised and do not appear in the graph. Thus, the known number of channels is a lower bound for the actual number of channels in the network. Regardless of

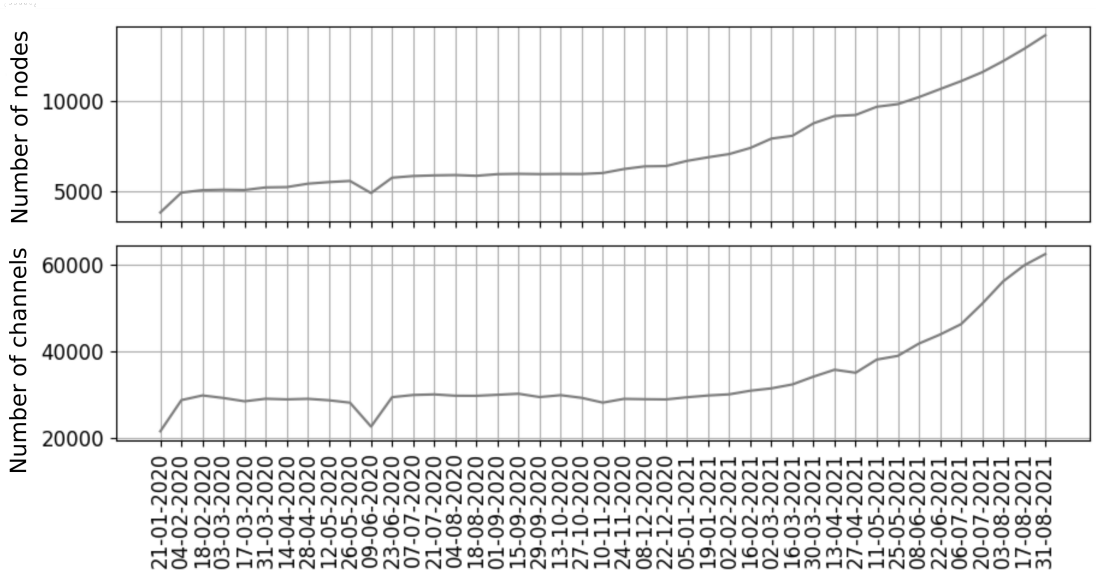

Figure 3.5: Evolution of the Lightning Network from January 2020 to August 2021.

channel status, channel balances are known only to the channel parties. The privacy of channel balances prevents external observers from tracking payments through channel balance monitoring. On the other hand, unknown balances challenge routing algorithms, which must find paths with sufficient funds to complete payments.

Intermediaries in Lightning charge two types of routing fees when forwarding payments: a base fee and a proportional fee[4] [85]. Base fees are a fixed amount charged to each forwarded payment. This fee financially stimulates nodes to route payments regardless of the payment value. Proportional fees, on the other hand, are charged as a percent of the amount to be forwarded. They are an extra incentive to route large payments, which lock more coins in the channel. The combination of both types of fees yields a unified routing fee defined as:

$$\phi_{ij} = \alpha_{ij} + \beta_{ij} HTLC_V, \tag{3.3}$$

where α and β are, respectively, the base fee and the proportional fee of channel (i, j), and $HTLC_V$ is the value of the HTLC to be forwarded. Despite featuring two types of routing fees, Lightning's fees are orders of magnitude lower than the fees charged for publishing a transaction on Bitcoin, making it especially attractive for real-time micropayments [86].

Lightning standardizes the message formats and protocols used in the network through the Basis of Lightning Technology (BOLT) [76]. BOLTs are documents inspired by Internet Requests For Comments (RFCs) that formally describe how developers should implement the network. Currently, the Lightning Network provides 11 BOLTs, which specify the format and exchange of messages for opening and closing

[4]The proportional fee is also named "fee rate" in the documentation of Lightning [76].

channels, payment encoding, routing protocols, and other specifications. The main Lightning Network implementations, Lightning Network Daemon (lnd) [4], Core Lightning (c-lightning) [5], and eclair [6], all follow BOLT specifications to ensure inter-platform compatibility.

Several works observe that the Lightning Network is centralized. Seres et al. [87] analyzed the topology of the Lightning Network in January 2019 and discovered a strong centralization tendency, with few nodes concentrating most channels in the network. The authors also analyze the robustness of the network, simulating attacks against the nodes with the highest degrees. They show that removing the top 37 nodes reduces the network capacity by half. Similarly, Lin et al. [88] assess the income concentration in the Lightning Network from January 2018 to July 2019. The authors verify a tendency of centralization around the higher degree nodes, forming core-periphery structures in which the core contains hubs and the periphery presents star-like substructures. The results show that removing the hubs partitions the network into multiple components, making it vulnerable to topology-based attacks [89].

3.5 PCNs: Other Networks

Although the Lightning Network is the default model of payment channel networks, other works have proposed PCNs with some differences to Lightning [20, 90, 91]. We highlight the most popular ones below.

Raiden. The most relevant alternative to the Lightning Network is Raiden, an off-chain payment channel network framework for the Ethereum blockchain [20]. Raiden, defined by its developers as "Ethereum's version of Bitcoin's Lightning Network", implements operations that are very similar to Lightning's. Firstly, the users must provide balance proofs, locking funds publicly in the blockchain. The assets to be locked are deposited in a smart contract that handles the Raiden transaction logic. After this phase, users can send payments for as long as they have sufficient funds. The payments are secured by an HTLC, as in Lightning. However, an advantage of Raiden over Lightning is the support for trading generic assets instead of only coins. It is possible, for example, to create a Raiden network per cryptocurrency or non-fungible token (NFT) that operates on top of Ethereum. There are nearly 15 thousand addresses that hold Raiden tokens, accounting for a total of $3.71M [92, 93]. Today's largest Raiden network is still under development and contains 128 nodes and 148 channels [94].

Sprites. Sprites [90] is a PCN and state channels proposal (recall that state channels are a generalization of payment channels) that differs from Lightning in two aspects:

(i) it proposes a coin-locking mechanism that replaces HTLCs, and (ii) it introduces partial withdrawals/deposits to/from payment channels. The new coin-locking mechanism of Sprites provides constant timeouts for coins locked along the payment chain, which speeds up coin unlocking in case some user needs to cancel the payment. The difference is that Lightning serially resolves HTLCs, one channel at a time, while the smart contract of Sprites can do this procedure concurrently. The notion of partial withdrawals/deposits in a channel allows users to allocate or remove funds on the fly without closing the channel. Thus, it reduces the need for costly on-chain operations. This feature is also possible due to the smart contract logic.

Pisa Sprites. Pisa improves Sprites by introducing a custodian service that allows users to disconnect for long periods without risking losing funds [95]. The custodian is similar to a watchtower service in Lightning, which we briefly explain in Section 4.4. Custodians alleviate the assumption that parties must always be online to guarantee security in the final state channel result. Despite providing interesting improvements compared with Lightning, Sprites and Pisa Sprites were not as influential in practice because they need complex smart contracts to implement the channel logic. The need for smart contracts restricts implementation to the Ethereum blockchain or other blockchains that support Turing-complete programming. The proposals also lack a large-scale implementation backed by the cryptocurrency community.

Teechain. Teechain [63, 96] is a PCN that leverages Trusted Execution Environments (TEE) to provide security for off-chain payments. Teechain allows disputes to be solved asynchronously instead of during a dispute period. To accomplish this, Teechain moves the root of trust from the blockchain to the TEE, ensuring that the nodes always act honestly. The architecture uses special CPU instructions provided by Intel's Software Guard Extensions (SGX) to create memory enclaves isolated from the operating system. In the proposed system, the client application maintains deposits within enclaves and updates the balances of the deposits when receiving or sending transactions through payment channels. The application with enclaves only interacts with the blockchain at deposit creation and completion. Before creating a deposit, the user must utilize an Intel attestation mechanism to ensure that the application runs in a genuinely trusted environment and will honestly follow the protocol. Furthermore, the system replicates data between device enclaves participating in a committee to prevent a single point of failure.

Teechain is compatible with the Bitcoin network but locks funds in the TEE instead of using HTLCs to transfer assets in the PCN. The remaining operations are similar to Lightning. The main drawback of Teechain is that it needs specific hardware. Thus, like proposals on the hardware layer, the adoption of this payment channel network is limited to use cases in which users can afford TEE-enabled devices.

Blind Off-chain Lightweight Transactions. BOLT [97] is a PCN proposed to guarantee anonymous payments, designed for ZCash [98] and other anonymous cryptocurrencies. Technologies such as pseudo-random functions and non-interactive zero-knowledge proofs enforce anonymity. BOLT has three types of channels: unidirectional, bidirectional, and indirect. Unidirectional channels allow consumers to pay with recurrence to a merchant, as in Spillman channels. Bidirectional channels enable recurrent payments between two parties but use zero-knowledge proofs instead of exchanging signed transactions. Finally, indirect channels are the abstraction for recurrent multi-hop payments between users that do not share a direct channel. In this case, users send payments through untrusted intermediaries using a revocation scheme similar to commitment transactions on Lightning but leveraging zero-knowledge proofs to hide the payment balance. Like other PCN alternatives, assessing BOLT's advantages and drawbacks in practice is challenging as there is no large-scale implementation for it.

Tumblebit. TumbleBit [99] is a unidirectional payment channel hub compatible with the Bitcoin protocol. TumbleBit maintains user privacy using an intermediary called Tumbler, which cannot link the payments between the parties involved in transactions. The Tumbler is a special user that mixes the received transactions with cryptographic techniques to ensure that the blockchain will not record any information about transactions between other users. Users can also use Tumblebit as a mixer to ensure privacy in non-recurrent transactions. Tumblebit is centralized and does not support payment forwarding through generic untrusted intermediaries.

3.6 Summary

Payment channels accelerate recurring transactions between a pair of users through direct rebalancing of previously-escrowed coins in the blockchain. Off-chain validation rules that provide valid state revocation and financial punishment for cheating parties that deviate from the protocol guarantee the security of payment channels. The detection of an attack is accomplished with constant monitoring of the blockchain to check if either party published an invalid balance. Constant monitoring implies that both parties must be online and have access to the blockchain while the channel is open.

Payment channel networks connect direct payment channels between users. A network of connected payment channels enables multi-hop payments through untrusted intermediaries, allowing users to send payments to destinations they cannot reach directly. Hashed Timelock Contracts ensure intermediaries cannot steal payments while the payments are in flight. Each intermediary in the payment path charges

a small routing fee as an incentive to route the payment honestly. Routing fees usually represent a tiny fraction of the payment value. Despite involving path-finding algorithms and routing fees, multi-hop off-chain payments are orders of magnitude faster and cheaper than payments in the blockchain.

Despite the existence of many PCN proposals, the Lightning Network is by far the most widely adopted and has become the default PCN. Other PCN proposals often lack large-scale implementations and community validation, hindering their adoption and gain of market share. The Lightning Network often incorporates several ideas from alternative PCNs, centralizing the development of a standard PCN model. Hence, many of the mechanisms involved in a PCN directly derive their functionalities from Lightning.

Chapter 4

Payment Channel Networks with Resource-Constrained Devices

In previous chapters, we hinted that some mechanisms of PCNs have limitations for specific use cases. This chapter shows how these limitations affect resource-constrained devices that often struggle to execute payment-securing procedures. The chapter aims to answer our first research challenges, *Challenge #1 (Architecture)* and Challenge #2 (Channel Security), by proposing an adapted PCN architecture and analyzing a critical vulnerability in payment channels formed by light nodes. We propose a countermeasure to the problem using minimum lock-time windows.

4.1 The Challenges of PCNs with Light Nodes

Recall from Section 3.2.3 that the security of a payment channel depends on mutual monitoring of the blockchain by channel parties. Also, recall that most PCN implementations adopt onion routing as the default payment transport protocol and that pathfinding algorithms need an updated graph topology to find paths [2, 20, 97]. Despite being fundamental to guarantee security and privacy to users, these mechanisms require nodes in a PCN to have the following:

1. **High availability.** Nodes need to stay online while the payment channel is open. They cannot disconnect for long periods, as this allows the other party to publish revoked channel states. Besides, they must constantly update the graph with the latest gossip messages.

2. **Enough bandwidth, storage, and computing capacities.** As users detect attacks through blockchain monitoring, the nodes must continuously download, store and verify blocks, which takes different resources. Besides, pathfinding and onion routing often involve computationally-heavy tasks that spend energy and must complete in reasonable times.

Nevertheless, resource-constrained mobile phones, smart objects, sensors, and others often lack both properties. These devices have low computing capabilities and can disconnect for days. Worse, executing heavy tasks can rapidly deplete the energy of battery-powered devices, pruning them from the network for an undetermined period. Hence, PCNs must adapt their procedures to support payments from light nodes with lossy connections and limited blockchain access.

4.2 Hybrid Payment Channel Networks (HPCNs)

We propose a hybrid PCN architecture composed of a reliable core and peripheral unreliable light devices with limited resources and wireless connections. We argue that a hybrid topology is the most impactful as it is the standard model for IoT platforms and mobile device architectures that rely on gateways and edge computing. Figure 4.1 depicts the topology of our network architecture. We consider two types of nodes:

- **Full nodes (FN)**, which compose the core network and act as payment routers. Full nodes may represent service companies, telcos, or any node with computing power to store a full copy of the blockchain. Full nodes are online with high probability and communicate via a reliable transport protocol. Although churn can occur in the core network, the probability that a full node quits the network without closing its payment channels is negligible compared with light nodes.
- **Light nodes (LN)**, which are resource-constrained devices that connect to the network via lossy wireless connections. Light nodes may represent mobile phones, sensors, smart objects, or IoT devices with limited computing and storage capacities. We assume light nodes can disconnect anytime without adequately closing the payment channel due to battery depletion, hardware malfunction, environmental conditions, or other reasons. We assume light nodes establish unreliable connections to send/receive blockchain transactions and request channel state verification. We also assume they can perform public-key cryptography to sign transactions but cannot store blocks.

We consider that full nodes connect to other full nodes via payment channels with a large capacity to route payments. Light nodes connect to one or more full nodes via minor and possibly unidirectional payment channels[1]. Henceforth we refer to channels between full nodes as *core payment channels* and between a light node and a full node as *edge payment channels*. We do not consider payment channels

[1]Note that "unidirectional channel" here means a bidirectional channel model in which all the balance is on one side, which prevents transactions from the other end. Not to be confused with one-way (Spillman) channels.

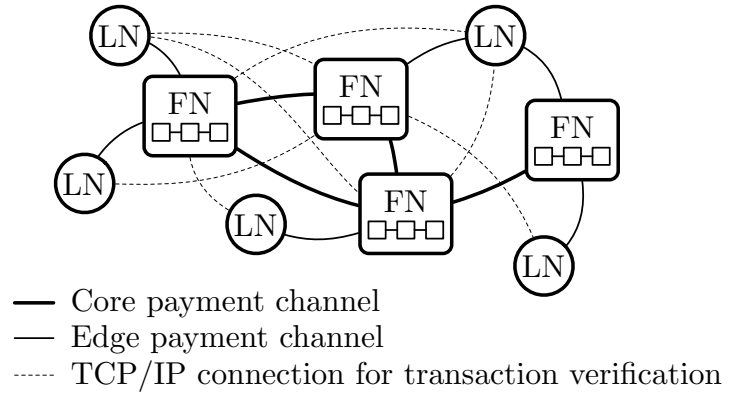

Figure 4.1: An example of a hybrid payment channel network. The light nodes (LN) represent wireless resource-constrained devices, and the full nodes (FN) represent capable nodes that store a copy of the blockchain. Light nodes establish TCP/IP connections to multiple full nodes to verify the states of their channels in the blockchain.

between light nodes as it would be unlikely for two resource-constrained devices to transact with each other for an extended period continuously. *Entry nodes* are the full nodes a light node selects to establish channels. Light nodes also establish TCP/IP connections to other full nodes to verify the states of their channels and avoid eclipse attacks. We name this architecture Hybrid Payment Channel Networks (HPCN) and define it below [100]:

Definition 3. *(Hybrid Payment Channel Network (HPCN)).* A hybrid payment channel network is a time-varying directed graph $G(t) = (V(t), E(t))$, where $V(t)$ is the set of devices in the network at time t and $E(t)$ is the set of payment channels that are open at time t. Any device $u \in G(t)$ can alter the set $E(t)$ of edges via three possible operations:

- `openChannel`$(\langle i, j\rangle, \langle B_i, B_j\rangle, W)$ opens a payment channel (i, j) with initial balances B_i and B_j. The window W defines a lock-time window where neither party can claim the coins if the channel is closed unilaterally. The operation publishes a transaction in the blockchain that must be signed by i and j;
- `closeChannel`$(\langle i, j\rangle, TX(t))$ closes the payment channel (i, j) with commitment transaction $TX(t)$, which contains the latest balance that has been signed at time t by both parties and publishes it in the blockchain. This operation can be issued cooperatively by having both signatures or unilaterally by some party. If the channel is closed unilaterally, the party that closed the channel can only claim his/her coins after the predefined time window W;
- `pay`$(\langle s, t\rangle, \pi, P_V)$ transfers a value of P_V coins from source s to target t via path $\pi = ((s, r_1), (r_1, r_2), ..., (r_n, t)) \in \mathbb{R}^n$. We assume the path π is defined by s before issuing the operation. Every hop from s to t will decrease its capacity by P_V coins in the direction of the payment's recipient if the whole path has enough capacity. Otherwise, the operation fails, and all channels remain unaltered.

4.3 The Coin Theft Problem in HPCNs

Current PCN implementations like Lightning [2] and Raiden [20] assume that any node transacting in the network remains online while the channel is open. This assumption mitigates the *coin theft problem*, in which one party publishes an old state to recover his/her sent coins as soon as the other party disconnects. As explained in Section 3.2.3, the system punishes attackers by allowing the victim to spend all coins in the channel if it recovers during the lock-time window. *Hence, it is only worth attempting the attack if the attacker can guarantee that the victim will not verify the blockchain until the time window expires.*

In our network architecture, however, light nodes can naturally disconnect for long periods or indefinitely. Device downtime is particularly challenging for use cases where the trend of payments is biased towards a light node, such as when a seller uses his/her device to receive transactions from multiple buyers or when a buyer uses

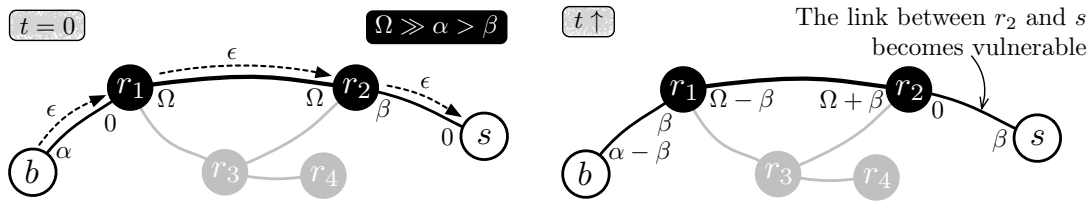

Figure 4.2: An example of the coin theft vulnerability in HPCNs. On the left, a continuous amount of ϵ coin flows from buyer b to seller s until it depletes the channel between r_2 and s. Then, on the right, s becomes highly vulnerable if it disconnects because r_2 has nothing to lose by closing the channel with a previous state.

his/her device to buy services from multiple sellers. We formulate the problem and demonstrate how imbalanced channels in resource-constrained environments enhance the probability of malicious behavior.

Let two resource-constrained devices, b and s, represent devices from a buyer and a seller, respectively, and be connected to entry nodes r_1 and r_2 via unidirectional payment channels as shown in Figure 4.2. Each payment channel (i, j) has an associated balance pair $B_{ij}(t) = (B_i(t), B_j(t))$, where $B_i(t)$ and $B_j(t)$ are the balances of nodes i and j at time t, respectively. For edge payment channels between buyers and entry nodes, e.g., (b, r_1), we assume an initial balance of $B_{br_1}(0) = (\alpha, 0)$, where α is the number of coins that buyer b reserves for payments in the channel. Conversely, the initial balance of edge payment channels between sellers and entry nodes, e.g., (r_2, s), is $B_{r_2s}(0) = (\beta, 0)$ where β is the number of coins the entry node r_2 reserves for routing payments to the seller s. We assume for simplicity and w.l.o.g. that s and b only participate in a single payment channel.

Once payment $\texttt{pay}(\langle b, s\rangle, \{b, r_1, r_2, s\}, \epsilon)$ occurs from b to s in this configuration, r_2 and s sign a commitment transaction $TX(1)$ containing the new balance of channel $B_{r_2s}(1) = (\beta - \epsilon, \epsilon)$. If s disconnects at this moment, r_2 can close the channel with the operation $\texttt{closeChannel}(\langle r_2, s\rangle, TX(0))$ and recover ϵ coins. Doing so is risky because r_2 would lose $\beta - \epsilon$ coins if s recovers before the lock-time window expires. However, as s receives more payments, the balance in (r_2, s) will converge to $B_{r_2s}(t) = (0, \beta)$. Once this happens, r_2 has nothing to lose by closing the channel with a previous transaction even if s recovers on time. Thus, attacking is the optimal strategy for any rational entry node once an edge payment channel to a seller is depleted. The seller is prone to coin theft even without actual malicious nodes. Malicious nodes may also attack intermediary cases once the victim disconnects if they expect a good risk-benefit ratio.

Note that the coin theft problem is unlikely to occur to senders because the other party can only lose coins by publishing an outdated state. For a generic situation in which light nodes act as senders and recipients, every light node becomes vulnerable

as soon as it receives coins. The efficiency of coin theft attacks also depends on the time window W set during channel-opening operations, which determines the maximum time a victim has to recover from the attack.

4.4 Defining a Minimum Time Window

One solution to the coin theft problem is to hire "watchtower" nodes to constantly verify the blockchain aiming to detect channels that have been improperly closed [2, 20, 101]. Watchtowers would receive the latest punishment transactions from the channels they monitor and charge a fee to issue them in the blockchain if needed. The solution, however, assumes watchtowers will act honestly, and it only works if the victim manages to send the punishment transaction to the watchtower before disconnecting. This cannot be guaranteed with wireless resource-constrained devices. Another countermeasure could be to create a reputation system for full nodes that identifies malicious agents. Reputation-based detection, however, is less efficient in anonymous environments such as PCNs. Besides, reputation is prone to centralization and attack vectors that are difficult to handle in decentralized environments [102].

Instead of adopting watchtowers or creating a reputation system, we propose a simple statistical approach: *discover a lock-time window W that minimizes the chance of attacks*. Since most PCNs already adopt default lock-time windows as a security measure [2, 20, 103], we believe this approach is the easiest to implement as it would require no modifications to PCN protocols. To the best of our knowledge, this is the first work to propose attack prevention through minimum time-lock windows. Our contribution also applies to traditional PCNs as a guideline for users to select the best window parameters based on their expected connectivity patterns.

Window size trade-off. The lock-time window, W, defines the time the closing party must wait until it can recover the coins. In other words, if a node on a channel disconnects and a coin theft attack occurs, the victim has W blocks to recover, verify the blockchain, and punish the attacker. Hence, the larger the size of W, the more secure the channel becomes. On the other hand, setting a W that is too large can create bottlenecks in routing and punish honest nodes that wish to close the channel correctly after the other party disconnects. In such cases, W should be as small as possible to improve coin liquidity and overall throughput. Therefore, W represents a trade-off between security and efficiency, and our goal is to minimize W while guaranteeing a minimum level of security.

Let s be a resource-constrained device. We propose a four-parameter methodology to estimate a minimum W:

(i) T_{off}, a random variable that models the time s remains disconnected from the

system, which can occur due to device failure or temporary signal loss. T_{off} can either be modeled through a continuous random distribution or be estimated with empirical data;

(ii) D_{det}, a random variable that models the delay for s to detect the attack. D_{det} follows a Poisson distribution with an expected value bound by the equation

$$E[D_{\text{det}}] = n\frac{E[T_{\text{off}}]}{b_t}\left(\frac{b_s}{d} + v\right), \tag{4.1}$$

where n is the number of full nodes from which s requests blocks, $E[T_{\text{off}}]$ is the average downtime of s, b_t is the block time[2], b_s is the average block size, d is the average download rate of s, v is the average time it takes for s to verify all transactions in a block. In this equation, $\frac{E[T_{\text{off}}]}{b_t}$ represents the number of lost blocks;

(iii) D_{pun}, a random variable that models the delay for s to punish malicious behavior after detecting it. As punishment incurs publishing a transaction in the blockchain, the distribution of D_{pun} follows the Poisson distribution found by Nakamoto [8] with expected value

$$E[D_{\text{pun}}] = n_c b_t, \tag{4.2}$$

where n_c is the number of blockchain confirmations it takes for a transaction to be considered valid, and b_t is the block time;

(iv) μ, a random variable that estimates the relative bias of each payment channel in the network. For empirical data, we calculate the bias of each channel in the dataset using the equation

$$\mu_{ij} = \left|\frac{B_i(0) - B_j(0)}{B_i(0) + B_j(0)}\right| \forall (i,j) \in E(t), \tag{4.3}$$

where $B_i(0)$ and $B_j(0)$ are the initial balances of each party in channel (i,j). Although the balances can arbitrarily change during channel operation, the bias μ_{ij} estimates how payments will flow since payments are more likely to occur from nodes with more capacity to nodes with less capacity.

Finally, each parameter composes the equation of W:

$$W = (T_{\text{off}} + D_{\text{det}} + D_{\text{pun}})(1 + \mu). \tag{4.4}$$

[2]The block time is the average time it takes for the consensus protocol to produce a block.

The rationale behind our definition is that the lock-time window must be at least $W = T_{\text{off}} + D_{\text{det}} + D_{\text{pun}}$; otherwise, the victim cannot recover and punish the attacker on time. Then, we proportionally increase the lock-time window by μ to improve the security of biased channels as we expect them to be more vulnerable. Note that because T_{off}, D_{det}, D_{pun}, and μ are either model distributions or real statistical data, W also yields a random distribution. The actual window size to be selected by a user depends on what level of security he/she wishes to adopt for his/her use case. Users who invest heavily in the channel should select higher thresholds to avoid great losses, and users who are willing to risk can select smaller thresholds to provide coin liquidity.

4.5 Proof-of-Concept Analysis

We use real data from the Lightning Network [2] and from mobile broadband connections [29, 30] in our prototype since they are the most widely adopted technologies for PCNs and light nodes [101, 104]. However, our proposed methodology is agnostic to blockchains, communication protocols, and PCN topologies. It suffices to estimate the parameters described in Section 4.4 to find a safe time-window value that addresses any specific use case. We provide the code of our implementation on GitHub[3].

4.5.1 Evaluation Setup

We create three scenarios based on real availability measurements of mobile broadband (MBB) devices to estimate the distribution of T_{off}. For the high-availability case, we use the downtime and packet loss distributions of MBB connections as measured by Elmokashfi et al. [30]. In their work, more than 90% of the connections use 4G technology, and the average downtime of a connection is 86.4s per day. The work from Baltrunas et al. [29] is a reference for the medium-availability case. The work measures the availability of mobile broadband connections that use 3G technology and shows that the downtime can last a few hours daily. Lastly, we simulate a low-availability scenario by shifting the medium-availability downtime distribution to the right by the average distance between the high-availability and medium-availability downtime distributions. This process yields an average downtime of about one week. By simulating three roughly symmetrical scenarios based on real data, we can predict how different levels of availability impact the minimum lock-time window. We could extend this approach to real-world device data of any kind.

For the detection delay D_{det}, we set the parameters as $n = 3$, $b_t = 600s$, and

[3]Available at https://github.com/gfrebello/pcn-time-window.

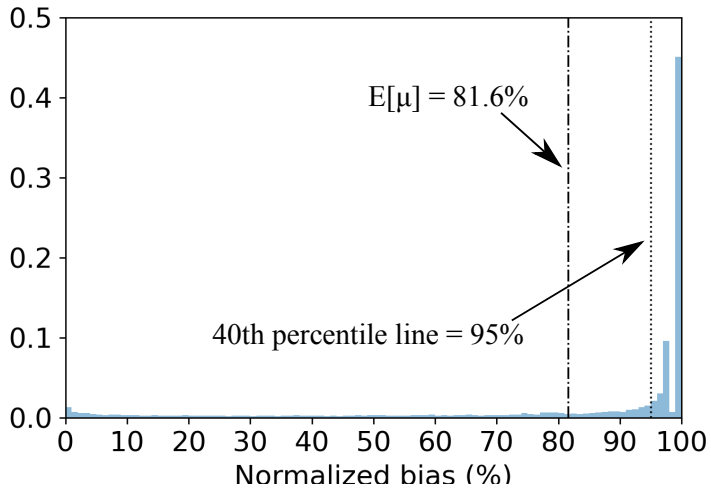

Figure 4.3: Normalized bias μ of payment channels in the Lightning Network. 60% of channels present over 95% bias towards one party, and the average bias is 81%, which indicates a heavily asymmetric behavior of payment flows.

$b_s = 10Mb/s$. $n = 3$ represents the minimum number of different nodes to request blocks to in case one node is faulty. b_s and b_t follow Bitcoin's average block time and block size, respectively. $E[T_{\text{off}}]$ is calculated according to the corresponding scenario, and the download rate d is set using previous MBB evaluations: $d = 30Mb/s$ for high availability, $d = 2Mb/s$ for medium availability, and $d = 1Mb/s$ for low availability [29]. The average number of confirmations $n_c = 6$ for the punishment delay D_{pun} follows the 6-confirmation rule proposed by Nakamoto in Bitcoin [8].

We extract the values of μ from LNChannels[4], an open-source tool that offers a data set of the Lightning Network. We download the channel balances from all closed channels since the beginning of the network and calculate the normalized bias μ_{ij} of each channel according to Equation 4.3. Figure 4.3 depicts the μ distribution.

Firstly, we observe a heavily asymmetric trend of payment flows, which confirms that the coin theft problem is not exclusive to HPCNs. Hence, adopting minimum lock-time windows that depend on channel bias may fit various PCN implementations. Secondly, the Lightning Network implementation causes a gap around the 99% percentile because it defines a minimum payment amount `dust_limit_satoshis` that, if not met, converts the transaction into channel fees [3]. This amount prevents parties from paying when the channel is almost empty.

Finally, we evaluate W through thresholds corresponding to the necessary value for a device to punish an attacker. A user that adopts $W(p)$ obtains p probability of recovering and assumes $(1 - p)$ probability of being attacked successfully. We use $W(50\%)$ as a reference for an unsafe threshold and $W(95\%)$ for a safe threshold and measure the W's trade-off by calculating the distance d between the two thresholds.

[4]Available at https://ln.fiatjaf.com/

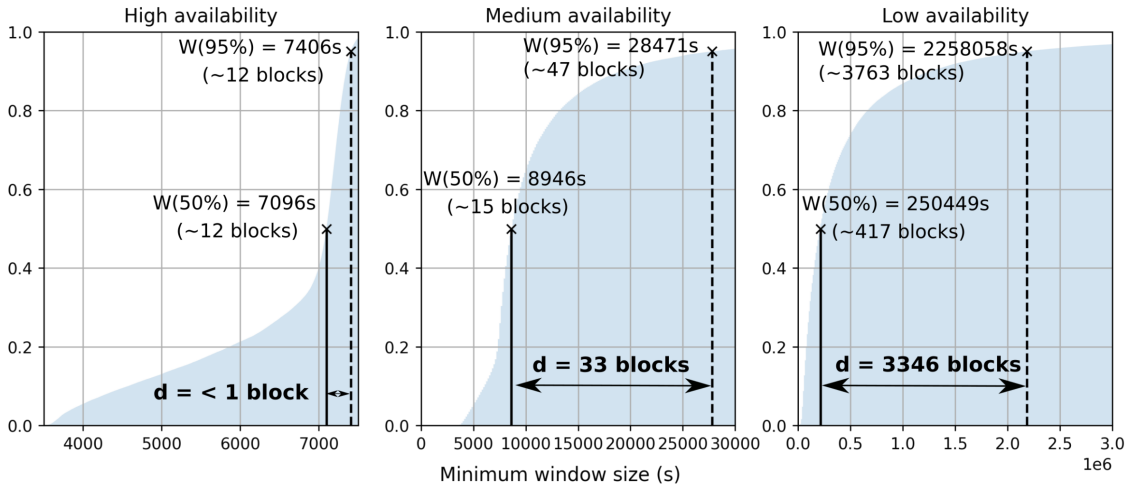

Figure 4.4: Lock time window sizes for all levels of availability with 6-block confirmation. When the availability is high, the distance d between the 50% and 95% thresholds remains below one block time, which indicates a small window is safe for most users. For medium and low availability, the distance increases significantly and forces the user to select a time window that better fits his/her security and delay needs.

Short distances mean no significant gain from adopting a smaller window size, while long distances mean the user should carefully select the value of W according to his/her needs. Figure 4.4 depicts the cumulative density functions for the minimum window sizes in all scenarios. The thresholds $W(p)$ are equivalent to the percentiles of the distribution of W.

4.5.2 Discussion

In the high-availability scenario of Figure 4.4, 4G connectivity allows devices to be safe from attacks even with short time windows. The distance of less than one block between $W(50\%)$ and $W(95\%)$ demonstrates that increasing W to a secure level generates no significant delay, so devices with good connectivity should adopt the safest W possible. The result also confirms that assuming good connectivity mitigates coin theft in traditional PCNs.

The trade-off between security and efficiency becomes significant in the medium-availability scenario. The safe threshold, $W(95\%)$, yields a dispute period of 28471s or approximately 8 hours. The distance of 33 blocks between $W(50\%)$ and $W(95\%)$ corresponds to an increase of 5.5 hours in return delays for the party that closes the channel, which happens because T_{off} is now the dominant parameter in Equation 4.4. The results indicate that a user with 3G connectivity should define minimum lock-time windows of at least a few hours to reduce the probability of attacks; otherwise, attackers with better connectivity can easily exploit them.

The low-availability scenario demonstrates that users with low connectivity should

either select W values in the range of weeks or use the blockchain directly to transact. Delays in such order of magnitude may be economically worthwhile if the fees to publish transactions in the blockchain are too expensive for the user. However, considering only time, more than 550 on-chain transactions could be sequentially published within the distance of 3346 blocks. This result indicates that the time window W may not be efficient for devices that plan to stay offline for extremely long periods. Instead, these devices should force-close their channels before disconnecting if possible and establish new channels when they reconnect.

4.6 Related Work

Several works propose adaptations of PCNs to include light nodes. Kurt et al. [105] propose LNGate, a lightweight protocol for IoT devices to use the Lightning Network [2] via untrusted gateways. Hannon et al. [25] propose a similar protocol and demonstrate its security and fairness using game theory. Robert et al. [26] propose an integration of the Lightning Network with existing large-scale IoT ecosystems. Mercan et al. [106] present alternative lightweight PCN implementations that focus on reducing the computational needs of mobile devices. Although some overlapping might occur, these works focus on adapting the Lightning Network to IoT scenarios while we propose a security mechanism agnostic to PCN implementations. Besides, we consider a simple PCN design that applies to any existing PCN.

Other works analyze the security of traditional PCNs, i.e., PCNs that require availability and resources from nodes. Tochner et al. formulate topology-based attacks that aim to disrupt the routing protocol of traditional PCNs [107]. Erdin et al. compare the security and privacy of several PCN implementations and identify emerging attack vectors [101]. The works neither discuss attacks in PCNs with resource-constrained devices nor present efficient countermeasures for wireless environments. To the best of our knowledge, our work is the first to propose an architecture for PCNs with resource-constrained devices, formulate the coin theft attack and propose a time window analysis as an efficient countermeasure.

4.7 Summary

In this chapter, we proposed a hybrid architecture that allows resource-constrained wireless nodes to issue off-chain transactions and analyzed the impact of the coin theft problem in such environments. Our main findings show that the problem is not exclusive to hybrid PCNs and that our solution may also work with traditional PCNs. A countermeasure based on minimum lock-time windows is efficient when the devices present high to medium availability. For devices with low availability, the

minimum lock-time window becomes so large that it may be better to close channels and publish the transactions directly in the blockchain.

Chapter 5

PCNSim: Payment Channel Network Simulation

Experiments with many devices and network topologies in real environments are often challenging to implement and analyze. A more practical approach is to test proposals through discrete-event simulations that provide controlled environments before deploying them into live networks [31, 108]. This chapter presents PCNsim, an open-source PCN simulator that creates and benchmarks payment channel networks under custom networking settings. PCNsim addresses Challenge #3 (Simulation) of the thesis, providing a realistic platform to simulate payments with different communication scenarios. PCNsim is available to the community at `https://github.com/gfrebello/pcnsim`. We compare PCNSim with the alternatives in the litetature in Section 5.4.

5.1 The Need for a PCN Simulator

Payment channel networks present open challenges regarding resource-constrained devices and other aspects that are difficult to address efficiently without flexible simulators [13–15, 59, 60]. For instance, comparing payment routing protocols usually involves benchmarking with numerous network topologies, payment sizes, and channel capacities. Analyzing traffic congestion requires sending numerous payments from different sources while monitoring channel balances. Measuring the minimum end-to-end latency of a payment demands no interference from other payments. Like datagram networks, these conditions are difficult to reproduce in practice but easy for discrete-event simulators [31, 108].

The absence of an automated tool that builds and analyzes PCNs forces researchers to adapt real PCN implementations to their specific use cases, which imposes several limitations. First, experiments must occur in small testnet topologies instead of

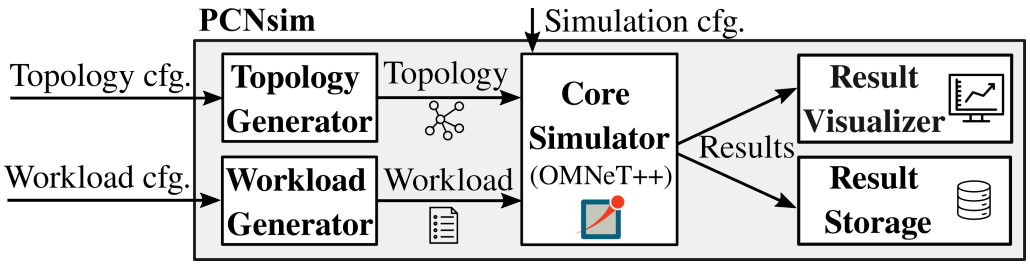

Figure 5.1: PCNsim's architecture. Users can easily customize topologies and workloads via configuration files.

large customized networks representing real instances. Second, payments cannot be sent from nodes the researcher does not control. Third, researchers cannot set channel capacities, and channel balances are private. Fourth, experimenting with resource-constrained devices is difficult since the default PCN software releases run full blockchain nodes that download the complete history of on-chain transactions. Finally, some implementations, such as the Lightning Network Daemon (LND) [4], are not optimized for high throughput, leading to high-latency transactions even in small networks [109]. The limitations above create the need for a platform that can simulate PCNs programmatically.

PCNsim. We propose PCNsim, a flexible, lightweight, and modular open-source payment channel network simulator. PCNsim includes topology and workload generator modules that allow researchers to automate their experiments on different scenarios through simple commands. The core of PCNsim extends the OMNeT++ [31] network simulator to reproduce the behavior of a PCN. As PCNsim runs on top of OMNeT++, the simulator also supports OMNeT++ extensions like INET [110] to build payment channels on top of wireless communication protocols and mobile networks. PCNSim follows the message format defined in the Lightning Network set of specifications [3] and allows users to model channel parameters, such as capacity and fees, based on real data collected from the Lightning Network [2].

5.2 PCNsim's Architecture

PCNsim is a simulator written in C++ language and OMNeT++ files, except for the generators, which we write in Python. It comprises five separate modules: (i) topology generator, (ii) workload generator, (iii) core simulator, (iv) result visualizer, and (v) result storage. We show PCNsim's architecture in Figure 5.1 and detail the main functionalities of each module below.

```
gabriel@localhost:~/opt/pcnsim/scripts$ python generate_topology_workload.py genTopo --help
Usage: generate_topology_workload.py genTopo [OPTIONS]

  Generates a topology for the simulation

Options:
  -t, --topology [scale-free|barabasi-albert|watts-strogatz]
                                  Topology used in the simulation
  -n, --nodes INTEGER             Number of nodes in the topology
  --alpha FLOAT                   Alpha parameter for scale-free topology
  --beta FLOAT                    Beta parameter for scale-free topology
  --gamma FLOAT                   Gamma parameter for scale-free topology
  -k INTEGER                      K parameter for Watts-Strogatz graph
  -p FLOAT                        P parameter for Watts-Strogatz graph
  -m INTEGER                      M parameter for Barabasi-Albert graph
  --lightning                     Channel capacities are modeled following
                                  real-world lightning network channels
  --help                          Show this message and exit.
gabriel@localhost:~/opt/pcnsim/scripts$
```

Figure 5.2: Topology generator. Users can create n-sized PCNs from random network models or a snapshot of the Lightning Network [2].

5.2.1 Topology Generator

The topology generator provides the network descriptor of a simulation round. It consists of a Python script that automatically generates a topology in the format the core simulator expects. The topology can be easily configured using user-set parameters from the command line, as shown in Figure 5.2. The module allows users to set the size of the graph and decide which network model to use for building payment channels. The current version of PCNsim supports parametrized random graph generation with Watts-Strogatz [111], Barabasi-Albert [112], and scale-free [113] models. The topology generator also supports modeling channel capacities and fees with random sampling from a real snapshot of the Lightning Network, which provides a realistic test environment.

5.2.2 Workload Generator

The workload generator creates a set of simulated payments. A payment in the simulations is a 4-tuple (s, t, P_V, T_S), where s is the payment's source, t is the payment's target, P_V is the payment value, and T_S is the payment's timestamp (at what time the simulator should issue it). The workload generator randomly samples s and t from a predefined graph's set of end hosts to create payment sets.

PCNsim supports two modes for workload generation. In predefined mode, the generator samples payment values randomly from a predefined range $[P_{Vmin}, P_{Vmax}]$. In dataset mode, it samples payment values from real-world datasets containing credit-card and e-commerce payments, which yields realistic results. Users can also define a limit for payment timestamps, forcing all payments to occur within a time window. Limiting the time is valuable, for example, to measure congestion in the

```
gabriel@localhost:~/opt/pcnsim/scripts$ python generate_topology_workload.py genWork --help
Usage: generate_topology_workload.py genWork [OPTIONS] FILE_LOC

  Generates a payment workload for the simulation

Options:
  --n_payments INTEGER   Number of payments in th network simulation
  --min_payment INTEGER  Minimum value of a payment in the network
  --max_payment INTEGER  Maximum value of a payment in the network
  --time_window INTEGER  Time window in ms
  --any_node             Transactions are issued by any node in the network,
                         not only end hosts
  --credit_card          Transactions are modeled following a credit card
                         dataset
  --e_commerce           Transactions are modeled following a e-commerce
                         dataset
  --help                 Show this message and exit.
gabriel@localhost:~/opt/pcnsim/scripts$
```

Figure 5.3: Workload generator. Users can create workloads of `n_payments` with random or sampled values from built-in data sets.

network. As with the topology generator, we provide a helper command containing brief descriptions of parameters, depicted in Figure 5.3.

5.2.3 Core Simulator

PCNsim's core simulator is the main component that replicates the behavior of a payment channel network and gathers user-set statistics. It implements the specifications of the Lightning Network's Basis of Lightning Technology (BOLTs) [76] into the OMNeT++ simulator [31], including the peer-to-peer messages involved in HTLC establishment and payment forwarding. Specifically, PCNsim implements the complete state machine of a standard channel operation using the message types defined in BOLT#3 [85]:

- `UPDATE_ADD_HTLC`: requests an HTLC with the target node;
- `UPDATE_FULFILL_HTLC`: claims an HTLC from the target node with a payment preimage;
- `UPDATE_FAIL_HTLC`: cancels an HTLC from the target node after an HTLC timeout or route failure;
- `COMMITMENT_SIGNED`: issues a new commitment transaction to the target node, requesting an update to the channel state;
- `REVOKE_AND_ACK`: acknowledges a received commitment transaction and signals revocation of the previous state.

The channel update protocol in PCNsim (and in LN) works in a 2-phase commit fashion to guarantee atomicity of state updates. A standard round of the protocol works as depicted in Figure 5.4. We detail each phase of the protocol below.

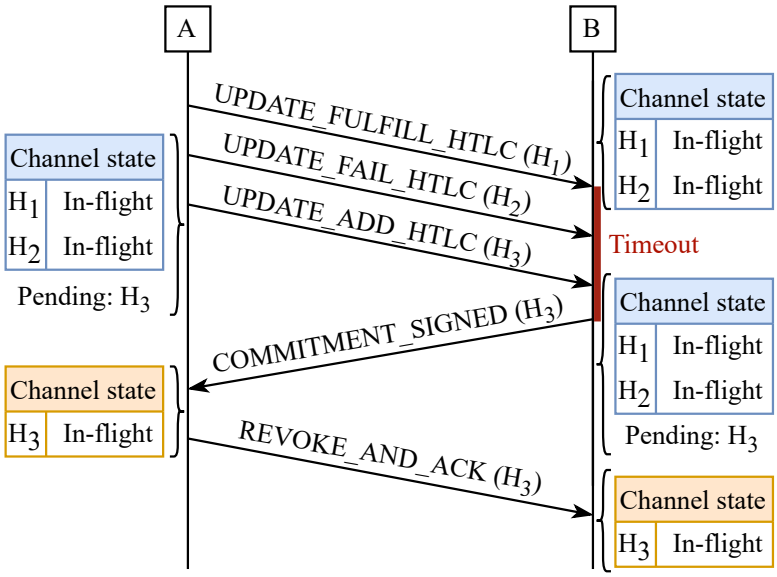

Figure 5.4: Sequence diagram of the channel update protocol implemented in PCNsim. The protocol follows the Lightning Network BOLT specifications [3]. Nodes start an internal timer when they receive the first HTLC change and trigger a new state commit when the timer expires.

To initiate or forward a payment with hash H_x in the channel (A, B), a node A crafts an HTLC which locks some coins in the channel and sends it within an `UPDATE_ADD_HTLC` message to B. A keeps the requested HTLC in memory with a pending state. Meanwhile, A can also request other state changes that cancel or claim an in-flight HTLC[1] through `UPDATE_FAIL_HTLC` or `UPDATE_FULFILL_HTLC` messages. Upon receiving an HTLC update message, B learns that A wants to update the state channel and places the HTLC in memory in a pending state. After a user-defined timeout, B crafts a new commitment transaction containing the pending HTLC changes, signs it and sends it to A through a `COMMITMENT_SIGNED` message. Note, however, that B does not update its channel state yet because it does not have a commitment transaction signed by A (recall from Section 3.2.3 that channel states should only be revoked with a commitment transaction signed by the other party). The `COMMITMENT_SIGNED` transaction contains B's revocation key for the current state. After verifying the message, A can safely commit the changes and update its channel state since it has a signed commitment transaction and a revocation key from B. Finally, A crafts a commitment transaction containing the new channel state and sends it to A with its key from the recently-revoked state. This message is named `REVOKE_AND_ACK` because it proves to B that A revoked the previous state and acknowledged the new one. Upon receiving `REVOKE_AND_ACK`, B can finally update its state, synchronizing it with the state in A and finishing the process.

[1] An in-flight HTLC is an HTLC that has been committed to the channel state and is waiting to be claimed or canceled.

We highlight that the above process follows precisely channel state updates in the Lightning Network. Thus, we can use it to model the expected latency of an HTLC confirmation. Let (i, j) be a payment channel, i be the node that requests the addition of an HTLC, and $X \in [0, T_j]$ be a random variable that models the time an HTLC waits in a pending state before the node sends the commitment transaction, given an internal timeout T_j set by j. We can model the delay between the sending of an HTLC request and its commit to the channel state on both nodes as

$$\delta_{ij} = 1.5RTT_{ij} + E[X], \tag{5.1}$$

where RTT_{ij} is the round-trip time of the channel as measured by i and $E[X]$ is the expected time HTLCs wait before the timeout. Note that this model assumes no messages are lost and that the time to construct the messages is negligible. For reference, the default timeout values in Core Lightning [5] and LND [4], two major lightning implementations, are 20 ms and 50 ms, respectively.

Apart from the messages involved in the channel update protocol, PCNsim adds two messages that help implement end-to-end payments:

- `INVOICE`: sends an invoice to the target node;
- `PAYMENT_REFUSED`: signals refusal to forward an HTLC;

The simulator uses `INVOICE` messages to transfer invoices from payment targets to payment sources. An invoice contains the payment hash H_x generated by the target, the target's address in the network, and hidden routes in case the target is behind private channels. When a payment source receives an `INVOICE` message, it computes a path to the destination and triggers the payment forwarding chain by requesting an HTLC with the first hop in the path. `PAYMENT_REFUSED` messages cancel pending HTLCs before committing to a new channel state. Nodes issue them immediately to the previous node in the payment path whenever they refuse to forward a payment.

Figure 5.5 depicts the peer-to-peer messages involved in a payment process. The process works as follows. The payment target, C, sends an `INVOICE` message containing one or more invoices to the payment source, A. A computes the payment path and sends `UPDATE_ADD_HTLC` to the first hop, B. B forwards the HTLC with another `UPDATE_ADD_HTLC` message in its channel with C. Note that if the payment had more than one hop, this would be done for each hop in the path until the HTLC reaches C. When the last `UPDATE_ADD_HTLC` message reaches C, C triggers the unlocking process by issuing an `UPDATE_FULFILL_HTLC` to the last hop, B. The hops propagate `UPDATE_FULFILL_HTLC` to the corresponding previous nodes, claiming the established HTLCs. The payment finishes when the last `UPDATE_FULFILL_HTLC`

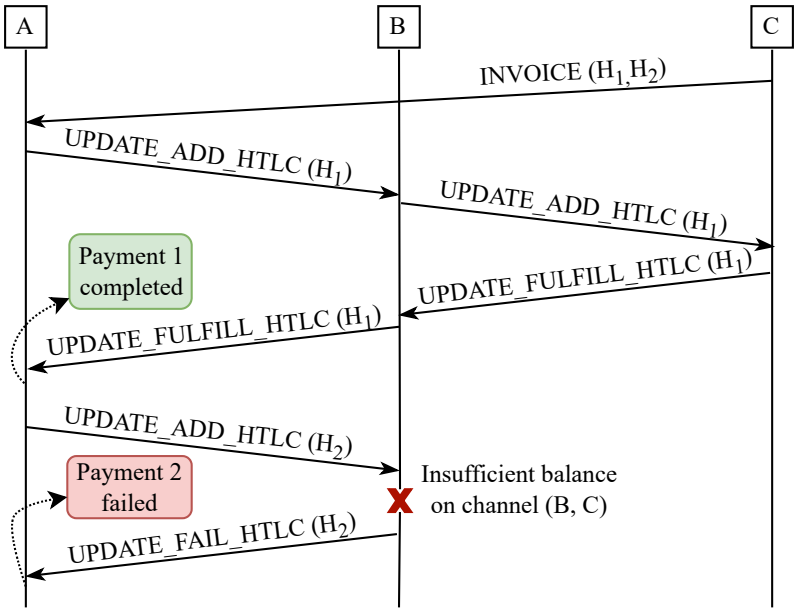

Figure 5.5: Sequence diagram of payments from a user A to a user C as implemented in the PCNsim simulator. We omit channel update messages for simplicity. The example shows a case where A only sends payment 2 after payment 1 to clarify the figure, but in reality (and in the simulator), payments can be sent concurrently.

reaches A, and the channel update is committed. If the payment fails during any step, a reverse sequence of `UPDATE_FAIL_HTLC` messages is triggered until all HTLCs are canceled.

We highlight that each HTLC in the above process involves channel state updates that must be committed. Moreover, an HTLC is only forwarded when committed in the previous channel [3]. For clarity, we omit these channel update messages in the figure, but they contribute to the end-to-end payment latency. Given a payment path $\pi = ((s, i_1), (i_1, i_2), ..., (i_n, t)) \in \mathbb{R}^n$, where s, t, and the i's are, respectively, the payment sender, the payment target and the intermediaries, the end-to-end latency of a payment that traverses π can be expressed by

$$L = \delta_{ti_n} + \sum_{(i,j)\in\pi} \delta_{ij}, \tag{5.2}$$

where δ_{ij} are the per-channel confirmation delays defined in Equation 5.1. The rationale for this model is that, for a payment to reach its destination, it must commit a sequence of HTLCs along the path, plus the target must unlock the last HTLC, claiming its coins. The rest of the unlocking process does not affect the payment latency, as the target already received what was due. Furthermore, the model assumes intermediaries take little time to forward HTLCs.

PCNsim's core simulator keeps HTLCs and channel states in memory rather than on disk, which enables fast payment processing and high-throughput payment

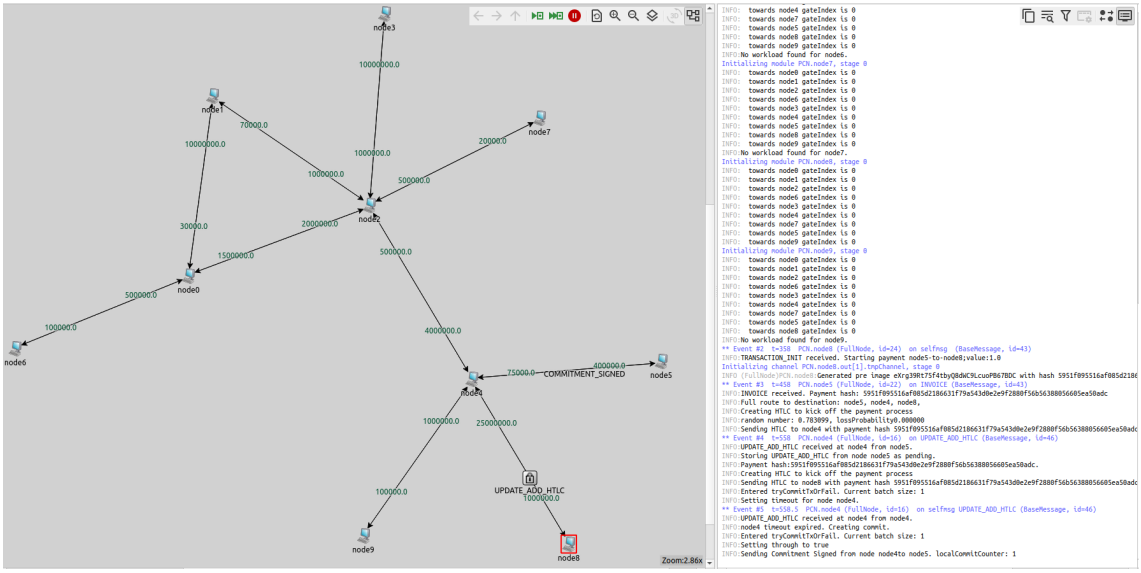

Figure 5.6: A graphical example of a PCN in our core simulator. The simulator displays logs for each node and channel in the network.

routing. We implement all messages in the application layer of OMNeT++ [31]. Developing a simulator that works in OMNeT++ removes the need for running a full blockchain node and provides lightweight software to experiment with. Besides, PCNsim allows users to implement PCN nodes as hosts with complete TCP/IP protocol stacks. The simulator supports OMNeT++ extensions like INET [110] that allow users to build payment channels on top of wireless communication protocols and mobile networks. The core simulator also provides simple debugging and monitoring of payment channels through log messages. We show a picture of PCNsim's graphical interface in Figure 5.6.

5.2.4 Result Visualizer and Storage

The result visualizer displays system statistics as gathered by the core simulator. It allows researchers to quickly analyze the impact of their proposals via automatically-generated results and graphs. The visualizer leverages built-in OMNeT++ tools to track PCN-related statistics, such as channel balances, payment latency, and average payment success rate. We illustrate an example where the visualizer is used to measure channel balance changes over time in Figure 5.7. Users can utilize the associated result storage module to persist the generated results into files in the disk for further analysis.

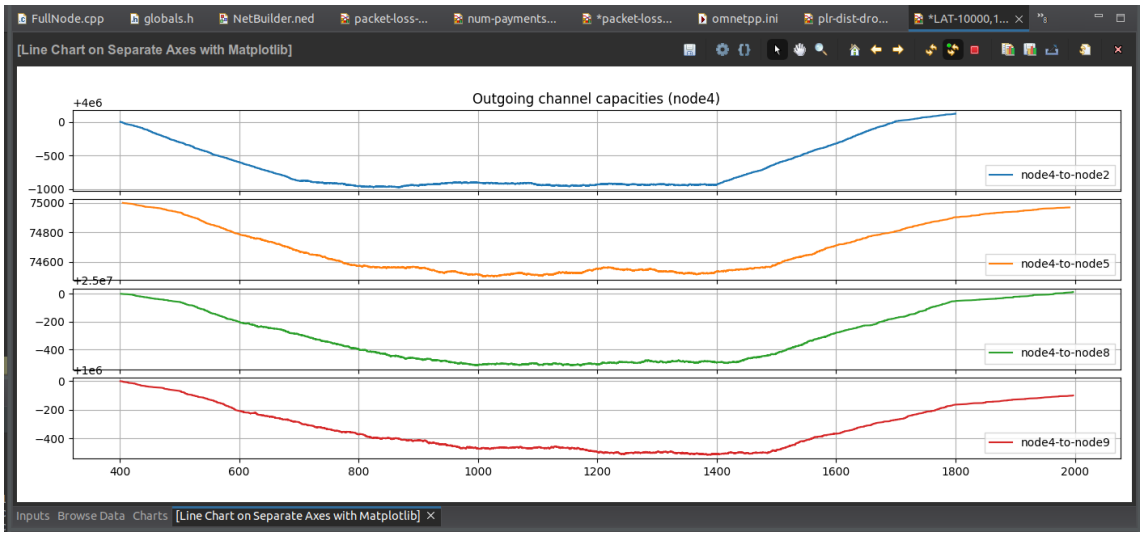

Figure 5.7: An example of PCNsim's result visualizer being used to monitor channel balances over time.

5.3 A Demonstration of PCNsim

We perform a few experiments in PCNsim to demonstrate how it can simulate several PCN attributes. Unless stated otherwise, our experiments simulate a 1,000-node scale-free network with channel capacities and routing fees sampled from the Lightning Network dataset [81]. We process a workload of 1,000 payments between random end hosts in the network.

5.3.1 Comparison of Payment Routing Methods

First, we compare two payment routing methods (R_M):

- `fee`: Lightning Network's fee-minimization algorithm, which is a Dijkstra's shortest path algorithm with routing fees as weights;
- `cap`: a variation of Dijsktra's shortest path algorithm in which the edge weights are the inverse of the channel capacities. The goal of `cap` is to maximize the chance that the payment will reach its destination by routing payments through high-capacity channels.

We compare the methods for different payment values (P_V): (i) small payments ($P_V = 10€$), (ii) large payments ($P_V = 200€$), and (iii) real credit-card payments[2] (average $P_V = 88.3€$). The objective of the comparison is to determine if routing methods that consider channel capacities provide a higher payment success rate, defined as

[2]Dataset available at https://www.kaggle.com/mlg-ulb/creditcardfraud.

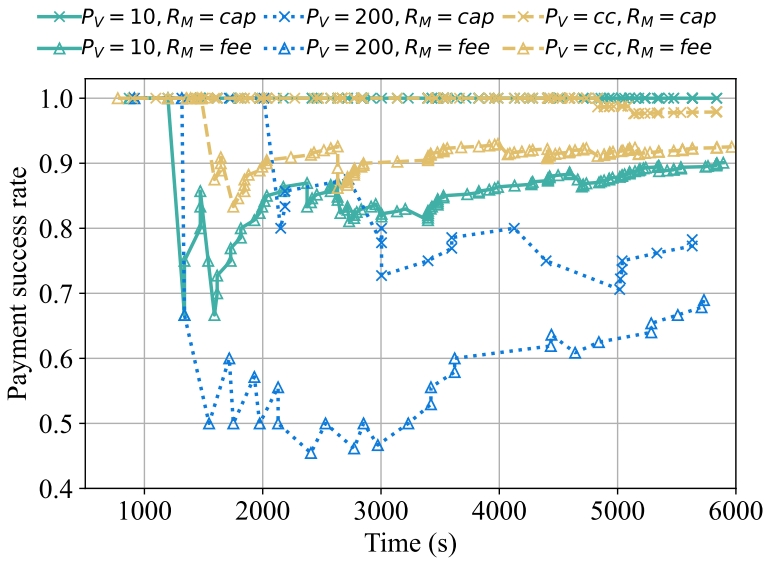

Figure 5.8: Payment success rate in our simulated PCN for several payment values (P_V) and routing methods (R_M). The results demonstrate that Dijkstra's shortest path approach with channel capacities as weights is more effective than the Lightning Network's fee minimization approach for all cases.

$$PSR = \frac{n_s}{n_f + n_s}, \tag{5.3}$$

where n_s is the amount of successfully-delivered payments, and n_f is the amount of payments that failed.

Figure 5.8 shows a time series of the payment success rates for all scenarios as measured by a random end-host. We observe two interesting findings. First, increasing the payment value impacts the payment success rate because large payments have a higher chance of failing when traversing a channel with a low balance in the path. Second, the capacity-based variation of Dijkstra's algorithm, `cap`, yields a higher payment success rate in all cases. It even compensates for the difference in payment values between credit-card payments and small payments. Hence, although minimizing fees reduces the cost for the end host, it incurs a higher chance of not completing the payment.

5.3.2 Payments Over Generic Communication Channels

Next, we send payments over a network where messages can be dropped. Specifically, we simulate a Hybrid Payment Channel Network, defined in Section 4.2, with resource-constrained devices as end-hosts. In this case, the communication channels between light nodes and their respective entry nodes are unreliable. We simulate payments between light nodes. Each payment traverses two unreliable edge payment channels:

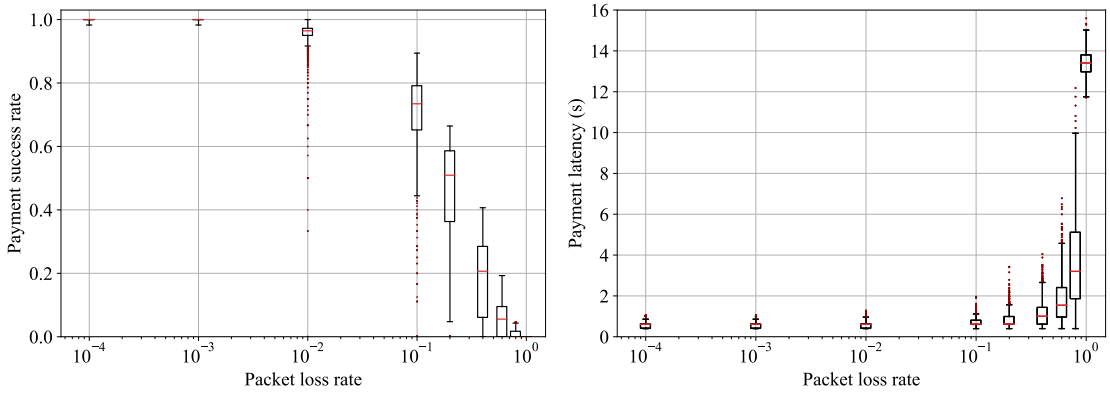

Figure 5.9: An analysis of two simulated transport protocols in PCNsim. On the left, the payment success rate degrades when sending payments over a UDP-like protocol that does not retransmit packets. On the right is the payment latency versus packet loss rate when adopting a TCP-like protocol that attempts to retransmit packets. The results show that unreliable communication channels impact the efficiency of payment channels.

one between the payment source and the first hop and another between the last hop and the payment target. Otherwise, the payment traverses the core network in which we assume payments cannot be dropped. We set the payment values to a minimum to guarantee payments fail exclusively due to packet drops in communication channels. For simplicity, we consider one payment is equivalent to one packet.

We analyze two scenarios corresponding to different transport protocols operating over unreliable communication channels. First, we simulate a UDP-like transport protocol that does not perform packet retransmission. In this case, packet drops immediately cause payment failures. Second, we simulate a TCP-like protocol that retransmits lost packets. We set the retransmission timeout to two round-trip times (RTT) and the maximum number of retries to 15, corresponding to the default values for TCP implementations in Linux distributions [114].

Figure 5.9 depicts the results for both scenarios. On the left, we show the impact of the packet loss rate on the payment success rate for the UDP-like protocol. The results demonstrate an evident degradation of the payment success rate when packet drop rates are significantly high. Likewise, the graph on the right demonstrates that packet drops cause extra retransmission delays in the TCP-like protocol. The end-to-end payment latency increases proportionally to the number of packet drops until it reaches a maximum value corresponding to the maximum number of retries. Both results illustrate that PCNsim can simulate payments and measure payment statistics under custom networking setups.

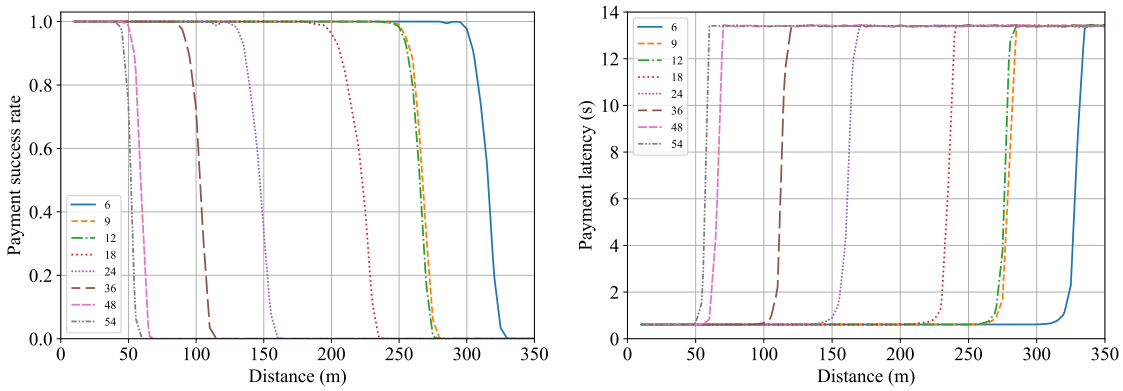

Figure 5.10: Payment success rate and latency of payments from resource-constrained devices connected via 802.11g wireless channels with different bitrates in Mb/s. When frame retransmission is deactivated, the success rate decreases with the distance from the entry node due to higher bit error rates. With retransmission, the average payment latency increases with the distance from the entry node.

5.3.3 Payments Over IEEE 802.11g Channels

Finally, we simulate payments over wireless connections with different bitrates [115, 116]. Payments, in this case, occur between two resource-constrained devices operating in 802.11g channels [117]. The devices are connected to their respective entry nodes in 802.11g ad-hoc mode at 10 mW transmission power. The bitrates of 6, 9, 12, 18, 24, 36, 48, and 54 Mb/s correspond to the Extended Rate Physical-layer (ERP) modes in 802.11g. Each bitrate yields a packet loss rate that varies with the distance from the device to the entry node. We analyze distances between 10 and 350 meters in 5-meter steps. As in the previous experiment, we consider one payment is equivalent to one packet.

In each simulation run, the payment source sends a single 56-byte payment to the entry node, which results in a 120-byte frame on the link layer. The same happens in the payment's last hop, except the packet goes from the entry node to the payment's target. At packet reception, OMNeT++'s error model computes the bit error rate from the signal-to-noise-plus-interference ratio, then calculates the corresponding packet loss rate. To make the model more realistic, the propagation model considers multipath propagation using two-ray ground reflections on flat ground. The devices are 1.5 meters above the ground. The simulation assumes isotropic background noise of -86 dBm. Similar to the experiment with transport protocols, here we consider the case where frame retransmission is deactivated and the default case where the protocol attempts to retransmit data frames at most 7 times.

Figure 5.10 shows the results. We omit the error bars to make the visualization clearer. The results show that the bit error rates increase with the distance between the device and the entry node, causing payment success rates to decrease for all

bitrates when retransmission is deactivated. Similarly, the average payment latency increases proportionally to the distance from the entry node when the protocol retransmits frames after a failure. The results also demonstrate that PCNsim correctly replicates 802.11g's rapid throughput loss when nodes transmit frames at a significant distance. The result is consistent with several previous works that analyze 802.11's performance in real environments [118–120].

5.4 Related Work

Payment channel networks are an emerging technology that needs large-scale deployment on real-world applications. As a consequence, there is a need for simulators that can effectively mimic the behavior of a PCN while allowing researchers to experiment with environments that represent potential applications. The absence of simulation tools for PCNs is a notorious issue that has been partially covered by other works, which we present below [109, 121–127].

CLoTH. CLoTH is a PCN simulator that produces performance measures such as the probability of payment success and the average payment latency [121]. Like PCNSim, CloTH aims to mimic the behavior of the Lightning Network according to its documented specifications. The current version of the simulator implements LN's default payment state machine, Dijkstra's shortest path routing protocols, and multipath payments. The main difference between CLoTH and PCNSim is that CLoTH simulates the Lightning Network in pure C language, whereas PCNSim operates on top of the OMNeT++ environment. Although this difference is insignificant regarding the implementation of Lightning Network functions, CLoTH's simulation is restricted to the application layer, while PCNSim can simulate the underlying communication network.

Spider. Spider [109], besides proposing a routing protocol, develops an event-based simulator for payment channel networks. The simulator extends the OMNeT++ simulation framework to model a PCN with Spider routers, providing the congestion control functionalities proposed in the paper. The code is open-source, and the authors use the simulator to extract statistics and compare their proposal with other routing protocols. The authors prove their simulation is sound by comparing the payment success rate of the simulator with a real Lightning Network implementation. The simulation, however, mainly focuses on Spider's routing protocol proposal while simplifying the core functionalities of PCNs, such as payment state control messages and HTLCs. Thus, using Spider as a generic simulator to test other payment strategies is difficult.

Blyskavka, LNSim, and LNTrafficSimulator. Other proposals implement simplifications of the Lightning Network. Piatkivskyi and Nowostawski [122] develop Blyskavka, a Lightning Network simulator in Java, to evaluate the impact of payment splitting when routing. Blyskavka simulates the Lightning Network operation rather than the Lightning Network itself, meaning it does not implement its specific messages and states. It also simplifies HTLC simulation by only blocking and releasing payments on the path after a short delay. Stasi et al. [123] develop LNSim, an LN simulator, to evaluate a novel fee definition and a multipath routing heuristic. Their open-source simulator can simulate the network at the LN protocol level but does not implement HTLCs. Beres et al. [124] develop LNTrafficSimulator, a Lightning Network traffic simulator based on LN public data, to analyze the economic and privacy implications of payments. Their work focuses on single-hop payments and simplifies other PCN functionalities.

Simulators for specific PCN functionalities. Several other proposals implement simple simulators for specific purposes. Kappos et al. [125] develop a PCN simulator in Python to evaluate whether an on-path adversary can successfully identify the payment sender. Their simulator uses publicly available Lightning Network snapshots and information published by central node owners, but the simulation code is yet to be published. CoinExpress [108, 126] develop a PCN simulation tool to test their routing proposal on top of the *ns-3* discrete-event network simulator. The simulator creates a random Watts-Strogatz network with random payments between users. The paper, however, does not clearly describe the simulator functionalities, and no source code is available. Papadis and Tassiulas [127] develop a discrete event simulator of a payment channel with support for transaction buffers. Their simulator aims to evaluate the impact of several single-hop payment forwarding strategies. Consequently, the simulator focuses on scheduling policies instead of providing a complete simulation of PCN functionalities.

Comparison of PCN Simulators. We highlight the main differences between the discussed simulators and ours in Table 5.1. Besides CoinExpress, Blyskavka, and Kappos et al., all simulators provide open-source code. Most simulators implement a version of the Lighting Network, with PCNSim and CLoTH being the only simulators that fully reproduce the phases involved in a payment process. However, to the best of our knowledge, PCNsim is the only simulator that provides a way to test PCNs with different underlying communication protocols.

5.5 Summary

Payment channel networks still present many open challenges that must be addressed through PCN simulation tools. We propose PCNsim, a PCN simulator that allows researchers to test new ideas intuitively and flexibly. Our demonstration shows that we can use the proposed system to efficiently compare different routing methods and measure their impact on the behavior of nodes and channels in a PCN. We also show that PCNsim can be used to send payments over unreliable communication channels that simulate wireless connections between resource-constrained devices and full nodes.

Table 5.1: Comparison between the existing PCN simulators in the litera

Reference	Language	Source code	Main features
CoinExpress [126]	C++/Python	Unpublished	- Based on the ns-3 network simulator - Implements several routing algorithms
LNSim [123]	C++	https://github.com/gdistasi/LNSim	- Simulates a simplified version of the LN - Generates networks via input or rando
Blyskavka [122]	Java	Unpublished	- Simulates a simplified version of the LN - Uses MASON as a simulation engine
Spider [109]	C++/Python	https://github.com/spider-pcn/spider_omnet	- Based on the OMNeT++ network simu - Implements routers with congestion co - Implements several routing algorithms - Generates networks via input or rando
LNTrafficSimulator [124]	Python	https://github.com/ferencberes/LNTrafficSimulator	- Simulates a simplified version of the LN - Generates traffic automatically based o LN snapshots - Generates networks via input or rando
CLoTH [121]	C/Python	https://github.com/marcono/cloth	- Accurately reproduces the LN specifications and code functions - Implements multi-path payments
Kappos et al. [125]	Python	Unpublished	- Simulates a simplified version of the LN - Focuses on PCN privacy guarantees - Supports graph inputs from LN snapsh
Papadis and Tassiulas [127]	Python	https://github.com/npapadis/payment-channel-scheduling	- Implements payment queues/buffers - Supports several single-hop payment scheduling policies
PCNSim [128]	C++/Python	https://github.com/gfrebello/pcnsim	- Based on the OMNeT++ network simu - Accurately reproduces the LN specifications and code functions

Chapter 6

Payment Routing with Resource-Constrained Devices

Despite being the most extensively studied topic of payment channel networks, payment routing still presents open challenges and opportunities for research, especially regarding support for light devices such as mobile phones, smart objects, and IoT sensors. In this chapter, we briefly introduce the background knowledge needed to understand the general problem of payment routing in PCNs, and then proceed to address the specific problem of routing with resource-constrained devices. We highlight that the chapter addresses Challenge #4 (Routing) and presents two main contributions: (i) a payment scheme that accelerates payment confirmations for light nodes, and (ii) an efficient pathfinding algorithm that considers application-specific constraints when computing paths.

6.1 Background on Payment Routing in PCNs

We mention in Section 3.3 that PCNs enable multi-hop payments that can be routed through a path using Hashed Timelock Contracts (HTLC). In a sense, this is similar to packets or messages being routed in a communication network. However, how PCNs deliver payments poses unique challenges that do not appear in classical datagram routing. We describe them in the following sections. Table 6.1 summarizes the most recurrent notations we use throughout this chapter.

6.1.1 Channel Capacities, Balances, and Liquidities

First, forwarding payments in a payment channel moves coins from the source to the destination, which reduces the source's ability to send new payments. Therefore, a channel's "forwarding capacity" depends on its initial fund distribution and how many payments have already traversed a specific direction. This particularity is

Table 6.1: Summary of recurrent notations used in this chapter.

Notation	Description
$\mathcal{N}(i)$	Set of i's neighbors.
$\mathcal{S}_w(i)$	Set of i's nearest neighbors sorted by metric w.
$\mathcal{W}$	Set of metrics.
$\mathcal{PCA}$	Path-cost-amount 3-tuple set.
$\hat{G}_l$	Estimated liquidity network.
R_f	Residual network of flow f.
ϕ_{ij}	Routing fee of arc (i, j).
Δ_{ij}	HTLC-resolution delay of arc (i, j).
p_{ij}	Probability of arc (i, j).
l_{ij}	Liquidity (arc capacity) of arc (i, j).
l_{ij}^{max}	Upper bound for the liquidity estimate in arc (i, j).
l_{ij}^{min}	Lower bound for the liquidity estimate in arc (i, j).
u_{ij}	Capacity of (undirected) payment channel (i, j).
π	Path of a payment.
π_w^*	Shortest path w.r.t. metric w.
f_{ij}	Flow on arc (i, j).
f_π	Flow on path π.
c_{ij}	Cost of arc (i, j) (single metric).
c_{ij}^w	Cost of arc (i, j) w.r.t. metric w.
$\vec{c}_{ij}$	Cost vector of arc (i, j).
c_π^w	Cost of path π w.r.t. metric w.
$\vec{c}_\pi$	Cost vector of path π.
$c(f)$	Total cost of flow f.
B^w	Upper bound for metric w.
$\vec{B}$	Bound vector.
$dist(i, j, w)$	Distance from i to j w.r.t. metric w.
σ	Main metric.
ψ	Constraint tightness.

the main difference of routing in PCNs compared to datagram networks, in which forwarding packets reduces the link's capacity only while packets are in transit. For instance, a 1 Gb/s link forwarding packets at 200 Mb/s in a packet-switched network has its capacity temporarily reduced to approximately 800 Mb/s, but the capacity returns to its original value once all the packets are delivered. In contrast, if a node A with a balance of 1,000 coins in a payment channel forwards 200 coins to a node B, the "forwarding capacity" in the direction from A to B is reduced to 800 coins indefinitely unless there is a payment from B to A. This characteristic highlights the need to keep channels balanced while routing or to adopt proactive channel rebalancing methods that attempt to stabilize channels when some party is running low on funds [129–131].

To formalize the problem, we replace the term "forwarding capacity" with the three concepts that precisely describe a channel's ability to forward payments: *capacity*, *balance*, and *liquidity*. The capacity of a channel is the aggregate amount put by the two parties into the 2-of-2 on-chain multisig address when they open the channel. It

represents the undirected maximum amount of value held in the channel by both parties and can be defined as:

Definition 5. *(Channel capacity).* The capacity u_{ij} of a payment channel (i, j) is the aggregate of the balances of the two channel parties, b_i and b_j, such that $u_{ij} = b_i + b_j$.

Note that the capacity u_{ij} of a channel is constant regardless of the current amounts held by each party inside the channel. In contrast to capacities, which are given per channel, channel balances are given per party and represent the maximum amount of coins that a party can send to the other end:

Definition 6. *(Channel balances).* The channel balances b_i and b_j of a payment channel (i, j) are, respectively, the amounts of coins held by i and j in the latest committed channel state.

Finally, the liquidities of a channel are the portions of channel balances that can actually be sent to the other party, given the current amount of escrowed coins. They take into account the in-flight HTLCs[1] in the channel, as they are supposed to modify channel balances when settled:

Definition 7. *(Channel liquidities).* Let η_{ij} and η_{ji} be, respectively, the total value of in-flight HTLCs from i to j and from j to i in channel (i, j). The liquidities l_{ij} and l_{ji} of the channel are, respectively, the available amounts that can be sent in the directions $i \to j$ and $j \to i$, such that $l_{ij} = b_i - \eta_{ij}$ and $l_{ji} = b_j - \eta_{ji}$.

The liquidities[2] in PCNs are the equivalent of a directed arc capacity in classical flow networks, which causes traditional max-flow approaches to inspire several payment routing protocols [132–134]. The main goal of such protocols is to select paths that have enough liquidity and are optimal according to some metric. Note that $l_{ij} \leq b_i \leq u_{ij}$ and that capacities, balances, and liquidities are non-negative integer units because they represent coins, i.e., $l_{ij}, b_i, u_{ij} \in \mathbb{N}_0$.

6.1.2 Uncertainty of Channel Liquidities

Another unique feature of PCNs is that only channel capacities are publicly available in the network [135]. Balances are not advertised systematically, and consequently, neither are channel liquidities. This preserves privacy and scalability in the system. First, announcing liquidities would allow payments to be tracked by statistical analysis of liquidity changes, compromising user privacy. Second, advertising liquidity updates

[1] Recall from the previous chapter that in-flight HTLCs are established but not yet claimed.

[2] We purposefully ignore Lightning's channel reserves in our liquidity definition as they are not standard in all PCNs [20, 63, 90].

for every channel and every state change would flood the network with messages, leading to scalability issues. Instead, channel balances and channel liquidities are kept private to the parties involved in the channel unless either party decides to disclose the channel state in an off-protocol manner.

The privacy of channel liquidities creates a significant challenge for payment routing: routing algorithms must compute paths based only on the information they know, which may yield paths that do not have enough liquidity to transport the payment. The efficiency of the routing protocol depends on its ability to estimate liquidities in the network correctly and to adapt to any liquidity information it may receive quickly. If this is incorrectly done, the algorithm selects supposedly feasible paths that fail when sending the payment. Then, the payment must be retried, increasing the user's latency and reducing the system's efficiency. In protocols that split the payment into multiple paths, a failure also compromises the atomicity of the payment. Failures in multipath payments can lead to situations where the buyer correctly pays n coins for a product, but the seller only receives $n - x$ coins, where x is the sum of the values of the failed payment parts.

Routing in PCNs involves, thus, a particular type of *transportation problem* in which arc capacities (channel liquidities) are uncertain [134, 135]. The liquidities can be in the interval $l_{ij} \in [0, u_{ij}]$ and must be estimated each time a path needs to be found. Namely, to start the process of pathfinding, a payment sender s must:

1. Build an undirected capacitated *channel graph* $G = (V, E)$ in which V and E are, respectively, the set of nodes and payment channels publicly announced in the PCN's peer-to-peer gossip protocol. The capacity of each edge $(i, j) \in E$ is the channel's advertised capacity u_{ij};

2. Build a directed capacitated *liquidity graph* $\hat{G}_l = (V, A)$ from G containing its complete node set V and replace each channel $(i, j) \in G$ with a pair of directed arcs $\{(i, j), (j, i)\} \in A$, such that $|A| = 2|E|$;

3. For each forward arc[3] $(i, j) \in A$, estimate the liquidity l_{ij} of (i, j) as a random variable $\hat{l}_{ij} \sim D[l_{ij}^{min}, l_{ij}^{max}]$, where D is some discrete probability distribution, $l_{ij}^{min} \geq 0$ is a lower bound of the arc's liquidity, and $l_{ij}^{max} \leq u_{ij}$ is an upper bound of the arc's liquidity;

4. For each backward arc $(j, i) \in A$, set the estimated liquidity of (j, i) to $\hat{l}_{ji} = u_{ij} - \hat{l}_{ij}$.

[3]We define the "forward" direction as the direction from the party who initiated the channel to the party who accepted the channel-initialization request. This information can be inferred from the fund distribution of the channel-opening transaction or be randomly set with a coin flip.

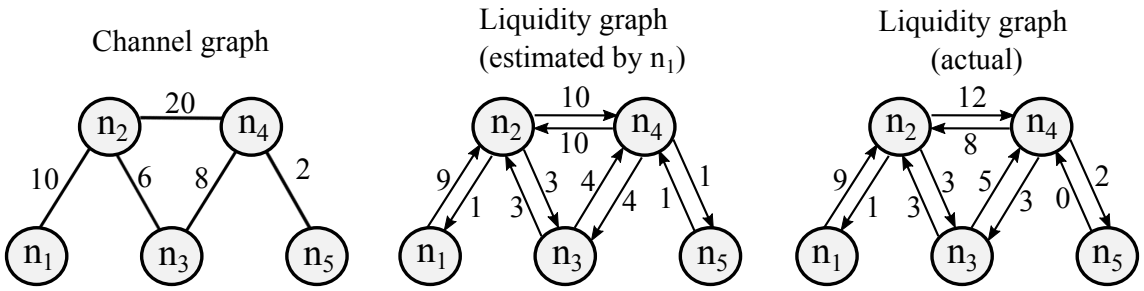

Figure 6.1: The graph structures involved in payment routing. The numbers represent edge capacities. Each node n_i transforms the public channel graph into a local liquidity graph that estimates the actual liquidities in the network.

Typically, the channel graph is built once at node start-up and updated every time a new node or channel announcement message is received. Thus, nodes must constantly listen for messages to keep the graph up-to-date. The estimated liquidity graph $\hat{G}_l$ is built once with arbitrary guesses at node start-up and updated every time the node obtains new information about channel liquidities (we explain how liquidity updates happen in Section 6.1.4). $\hat{G}_l$ represents a guess by the node of the liquidity graph G_l, which contains the actual liquidity distribution in the network and cannot be obtained. Note that the node does not have to guess the liquidities of channels in which it participates. We illustrate the difference between the channel graph, the estimated liquidity graph, and the actual liquidity graph in Figure 6.1.

6.1.3 Routing Payments through Minimum-cost Flows

As in many transportation problems, each arc in a liquidity graph has an associated weight w_{ij} that defines how much it costs to traverse it. Depending on how the graph is built, the weight can either be a scalar or a multidimensional tuple $w_{ij} \in \mathbb{R}^n$, where each dimension represents a different type of cost. For instance, arc weights in the Lightning network can be seen as 3-dimensional tuples $w_{ij} = (\phi_{ij}, \Delta_{ij}, p_{ij})$, where ϕ_{ij}, Δ_{ij}, and p_{ij} are, respectively, the routing fee, the minimum timeout delta, and the *a priori* probability[4] of arc (i, j).

The cost of an arc, c_{ij}, can be computed with a cost function $f : \mathbb{R}^n \to \mathbb{R}$ that maps arc weights into a scalar value such that $c_{ij} = f(w_{ij})$. In the simplest case, the arc costs are $c_{ij} = w_{ij}$ if w_{ij} is a scalar or $c_{ij} = w_{ij}^k$ if w_{ij} is a vector, where w_{ij}^k represents the k-th dimension of w_{ij}. However, cost functions can be arbitrarily complex as long as they are convex, to allow payment routing protocols to select paths based on cost-minimization algorithms [134]. As an example, the cost function in Lightning's default routing protocol is $c_{ij} = \phi_{ij} + \rho_1 \Delta_{ij} + \rho_2 p_{ij}^{-1}$, where ρ_1 and ρ_2 are user-set parameters that bias the cost towards a preferred metric.

[4]The *a priori* probability of an arc in Lightning is a guess of how likely a payment is to successfully traverse the arc before any attempt. It accounts for liquidity uncertainties.

With a defined cost for each arc, the problem of sending payments in a PCN becomes a single-source single-sink instance of the *minimum-cost flow*[5](MCF) problem in transportation networks. In this problem, we must transport a demand d from a source s to a sink t while minimizing the total cost of some *feasible flow*. The demand d corresponds to the payment value mentioned in previous chapters. Furthermore, flows in PCNs are integer because coins are indivisible units. We can formally define a feasible integer flow as:

Definition 8. *(Feasible integer flow).* A feasible integer flow between a source s and a sink t in a graph $G_l = (V, A)$ with liquidities $l_{ij} \in \mathbb{N}_0 \; \forall (i,j) \in A$ is a function $f : A \to \mathbb{N}_0$ such that:

1. $0 \leq f(i,j) \leq l_{ij} \quad \forall (i,j) \in A$ (capacity constraint),
2. $\sum_{(i,j)\in A} f(i,j) - \sum_{(j,i)\in A} f(j,i) = 0 \quad \forall i \in V \setminus \{s,t\}$ (flow conservation).

A feasible flow in a transportation network represents a mapping that allocates a portion of the demand into each arc (i,j). Each allocation must respect the arc's capacity (liquidity) and yields a per-arc cost given by the cost function c_{ij}. Note that, because of the flow conservation constraint, we can dissect a flow into separate paths that transport, each one, a fraction of the demand (see a proof for this statement in Appendix A.1). Thus, a flow f can always be rewritten as a 2-tuple set $f = \{(\pi_k, f_k) \mid f_k \leq d \quad \forall k \in \{1, ..., n\}\}$, where n is the number of paths in the flow and π_k is the k-th path, which transports f_k flow units from s to t. The cost of a feasible flow can be expressed per path as $c(f) = \sum_{\pi_k \in f} c(f_k, \pi_k)$, where $c(f_k, \pi_k)$ is the cost of sending f_k flow units through path π_k.

Generalized flows. We highlight that the flow model used in this formulation is a *conservative* flow, i.e., a flow in which the demand sent at the source s equals the demand received at the sink t. It ignores that nodes take a small fraction of the demand as a fee when routing payments (recall how HTLCs are chained in Section 3.3.1). A generalized flow model in which flows can be lossy better describes this behavior. The model replaces the flow conservation constraint by

$$\sum_{(i,j)\in A} (1 - \phi_{ij}) f(i,j) - \sum_{(j,i)\in A} f(j,i) = 0 \quad \forall i \in V \setminus \{s,t\}, \tag{6.1}$$

where ϕ_{ij} is the routing fee charged by i to forward the demand on arc (i,j). In the generalized flow formulation, d units of flow sent along an edge (i,j) become $d(1 - \phi_{ij})$ units of flow when they arrive at node j.

The generalized flow model precisely reflects how payments traverse channels in a

[5]Also known as a maximum flow with minimum cost.

PCN. However, the computational effort needed to solve minimum-cost flow problems with lossy flows is high and yields little benefit since routing fees typically represent an insignificant fraction of the demand. Instead, most routing proposals prefer to model the problem with conservative flows and add the routing fees to the demand before routing, which provides a good approximation of the generalized minimum-cost flow problem [4, 132–134, 136, 137]. Though we recognize that this is an approximation, we adopt the same approach as it yields good results in practice [103, 132, 134, 137–140]. Henceforth, we treat the problem of payment routing as a minimum-cost flow problem with flows as defined in Definition 8.

The goal of the MCF problem with conservative flows is to provide a feasible flow with the minimum total cost as measured by the cost function. As we can separate flows into paths, the objective function of the problem can be expressed by

$$\min_{f} c(f) = \min_{\pi_k} \sum_{\pi_k \in f} c(f_k, \pi_k), \tag{6.2}$$

where $c(f)$ is the cost of a flow f and $c(f_k, \pi_k)$ is the cost of sending f_k units along path $\pi_k \in f$. Assuming the path costs $c(f_k, \pi_k)$ are computed using additive metrics, i.e., $c(f_k, \pi_k) = \sum_{(i,j)\in\pi_k} c_{ij} f_{ij}$, where f_{ij} is the flow on edge (i, j) and c_{ij} is the cost of arc (i, j), we can define the per-arc formulation of the MCF problem with integer flows for some liquidity network $G_l = (V, A)$:

Definition 9. *(Minimum-cost flow problem).* Given a transportation network $G_l = (V, A)$ with liquidities l_{ij} $\forall (i, j) \in A$, arc costs c_{ij} $\forall (i, j) \in A$ and a demand d to be transferred from a source node s to a sink node t, the (integer) minimum cost-flow problem is an optimization problem in the form:

$$\min_{f_{ij}\in\mathbb{N}_0} \quad \sum_{(i,j)\in A} c_{ij} f_{ij}, \tag{6.3a}$$

$$\text{subject to:} \quad 0 \leq f_{ij} \leq l_{ij} \qquad \forall (i, j) \in A, \tag{6.3b}$$

$$\sum_{(i,j)\in A} f_{ij} - \sum_{(j,i)\in A} f_{ji} = \begin{cases} d, \text{if } i = s \\ -d, \text{if } i = t \\ 0, \text{otherwise} \end{cases} \qquad \forall i \in V. \tag{6.3c}$$

Because the minimum-cost flow problem is well-known in the literature [141], formulating the pathfinding problem for payments as an MCF instance allows us to use general-purpose solvers and classical optimization algorithms to compute payment paths [142, 143]. Note that the objective function in Equation 6.3a can be rewritten as

$$\min_{f_{ij} \in \mathbb{N}_0} \sum_{\pi_k \in f} \sum_{(i,j) \in \pi_k} c_{ij} f_{ij}, \tag{6.4}$$

which indicates that the MCF problem can be solved by selecting a sequence of paths that can transport as many units as possible at a minimum cost. Thus, when all the demand fits in a single path, the problem reduces to the shortest pathfinding problem, which can be solved using classical techniques. This is the main reason why most payment routing algorithms adopt some variation of Dijkstra's [144] or Bellman-Ford's [145] pathfinding algorithms [4–6, 20].

6.1.4 Liquidity Updates and Channel Probing

We mentioned in Section 6.1.2 that payment routing involves finding paths in a graph of uncertain liquidities. Routing algorithms must update the estimated liquidity network as best as possible before attempting to solve the MCF problem. Fortunately, this does not need to be done with purely-blind guesses because each payment attempt returns some information about channel liquidities.

Let a single-path payment attempt be a 4-tuple (s, t, d, π) that transfers d coins from source s to sink t through a path π. Payment attempts yield the following information when executed:

- If the payment fails to traverse an arc $(i, j) \in \pi$ [6], we can update the upper bound of (i, j)'s liquidity to $l_{ij}^{max} = d$, such that $\hat{l}_{ij} \in [0, d]$. We can also update the lower bounds of all arcs (u, v) before (i, j) in π to $l_{uv}^{min} = d$ as the payment would have traversed them if it had not failed in (i, j);

- If the payment successfully reaches t, we can update the upper bounds for each arc $(i, j) \in \pi$ to $l_{ij}^{max} = l_{ij}^{max} - d$ as we know the liquidity moved to the opposite arc;

- For each updated arc (i, j), we can recompute the estimates of the opposite arcs (j, i) using the channel's capacity like we did when first building the liquidity graph: $\hat{l}_{ji} = u_{ij} - \hat{l}_{ij}$.

Consequently, each payment attempt in PCNs improves the accuracy of liquidity estimates. A straightforward implication of this statement is that the more payments a node attempts to make, the more it knows about the current liquidity payment channels in the payment path. Moreover, the accuracy of liquidity estimates degrades with time because payments from other nodes are likely to happen. Some routing

[6] The information of which arc failed is given by default in all major PCN implementations [4–6].

proposals adopt age functions that adjust liquidity estimates according to the elapsed time since the last update [4, 132, 136].

Finding a suitable path in PCNs is, thus, not a single instance of the MCF problem; instead, it involves a trial-and-error approach that generates several MCF instances. At each payment attempt, the routing algorithm solves an MCF problem in the estimated liquidity graph, attempts to send the payment through the minimum-cost paths, and adjusts liquidity estimates according to the results. If a payment part (or the whole payment) fails, the algorithm can abandon it or try again, considering the newly obtained information. Some proposals proactively probe payment channels to provide an accurate estimate of liquidities when computing paths as a way to reduce the extra latency caused by payment retries [132, 136, 146]. In fact, as the payment source computes paths, any node can quickly disclose the liquidity of a target channel by sending fake payments to itself that include the channel into the payment's path [147–149].

6.2 Routing Payments from Light Nodes

Apart from the challenges of payment routing in general, resource-constrained devices such as mobile phones, smart objects, and sensors present additional limitations that hinder the adoption of common payment routing algorithms. First, these devices may disconnect frequently or for long periods, which affects their ability to update the channel graph and, consequently, to send and receive payments. Second, resource-constrained devices cannot store many blocks, forcing them to rely on full nodes to confirm channel updates. Finally, a resource-constrained device may not have enough computational power to execute complex pathfinding algorithms, which can affect the efficiency and accuracy of payment routing.

The state-of-the-art on routing algorithms for PCNs often ignores the limitations of light nodes. For instance, most routing proposals assume nodes always listen to channel update messages to synchronize and store a copy of the channel graph [109, 132–134, 138]. The main implementations of the Lightning Network and the Raiden Network further require nodes to store blocks to verify channel states and adopt onion routing to provide payment privacy at the expense of extra computational costs [2, 20, 22]. Such assumptions make it difficult for devices with limited resources and intermittent connectivity patterns to route payments as full nodes would, and little research has been done to change this scenario even though resource-constrained devices account for over half of all the traffic on the Internet [27, 150, 151].

Besides, many payment applications need minimum latency to work properly or consider it desirable. Namely, real-time payments are critical for stock markets [152], cross-chain trades [153], and other applications that operate under strict

time frames [154–156]. Sending payments from light nodes can be especially challenging in such cases, as the scarcity of resources adds extra latency to the payment process. Using normal payment schemes, devices that stay offline most of the time must download the latest link states whenever they want to send a payment or risk sending it with an outdated network view. These limitations create the need for a novel payment scheme that reduces confirmation latency and computational efforts when issuing payments from resource-constrained devices.

In the previous section, we focused on the problem of finding paths in a liquidity network. However, payment delivery in PCNs depends on multiple phases that we detail now to explain the rationale behind our approach:

1. **Invoice generation and forwarding.** The payment recipient generates an invoice containing the number of coins to be paid and their address in the network. It forwards the invoice to the payment sender using some arbitrary communication channel.

2. **Pathfinding.** The sender receives the invoice and attempts to find one or more feasible paths to the payment destination. Pathfinding typically uses classical shortest pathfinders or minimum-cost flow solvers on top of the estimated liquidity network.

3. **Route locking.** With a defined path (or path set), the sender establishes the first HTLC, triggering the sequential locking of the required funds along the route. Each intermediary establishes an HTLC with the next hop in the route after receiving an HTLC from the previous hop. The locking process stops when the last HTLC reaches the recipient. This process can be done via onion packets that only reveal the payment value and the next hop to any intermediary in the path.

4. **Route unlocking or payment settling.** When the last HTLC along the path is established, the recipient redeems it and starts the backward unlocking of the path. Each intermediary redeems the established HTLCs with the corresponding previous hop. The payment finishes when the first intermediary redeems the initial HTLC, which the sender established.

During invoice generation and transmission, all processing occurs on the recipient side while the sender waits. Route locking/unlocking follows a decentralized process that depends on the processing speeds and communication latency of hops along the route. The only actions a payment sender must perform in these phases are receiving the invoice and establishing the first HTLC along each route. On the other hand, pathfinding can be computationally expensive for the sender as it involves computing

a feasible path over a large graph. Moreover, keeping the network topology up-to-date requires constant connectivity to receive gossip messages from other nodes. This is easy for powerful servers or desktops but can become extremely costly and slow for nodes with low resources. Light nodes that disconnect for long periods cannot maintain the graph efficiently, and even if they had good availability, it would require extensive memory and battery usage.

In light of the characteristics above, we identify a clear direction for providing payments from resource-constrained devices in time-sensitive applications: *offload pathfinding to capable servers while accelerating the other phases as much as possible.* By doing so, not only do we alleviate the burden of light nodes, but we also reduce the amount of time they need to stay connected. Besides, this approach permits separating the problem into two sub-challenges: (i) how to speed up payment confirmation for light-node payments, and (ii) how to efficiently find paths in full nodes under application-specific constraints. We address each challenge separately in the following sections.

6.3 Accelerating Payment Confirmations

The first part of the problem is to ensure the recipient obtains a payment confirmation as soon as possible. Suppose the payment follows the normal process described in the previous section. In that case, payment confirmation happens when the last HTLC in the payment path is committed, which guarantees to the recipient that the sender found a feasible path and all HTLCs were successfully established. As payment confirmations occur per path, the recipient must wait for the confirmation of the last HTLC on each path in a multipath payment. We can quantify the confirmation latency of a payment part k that traverses a predefined path $\pi_k = ((s, i_1), (i_1, i_2), ..., (i_n, t)) \in \mathbb{R}^n$ from source s to destination t as:

$$\lambda_k = \sum_{(i,j)\in\pi_k} \delta_{ij}, \tag{6.5}$$

where δ_{ij} are the per-channel state update delays as defined in Equation 5.1. The confirmation latency of a payment part is similar to the end-to-end single-path payment latency defined in Equation 5.2, except that it does not include the target's claim of the last HTLC. The confirmation latency of a multipath payment f composed of single-path payment parts k, can thus be expressed by:

$$\Lambda(f) = \lambda_{inv} + \lambda_{MCF} + \max(\lambda_k \ \forall k \in f), \tag{6.6}$$

where λ_{inv} is the time it takes to transfer the payment's invoice from the recipient t to

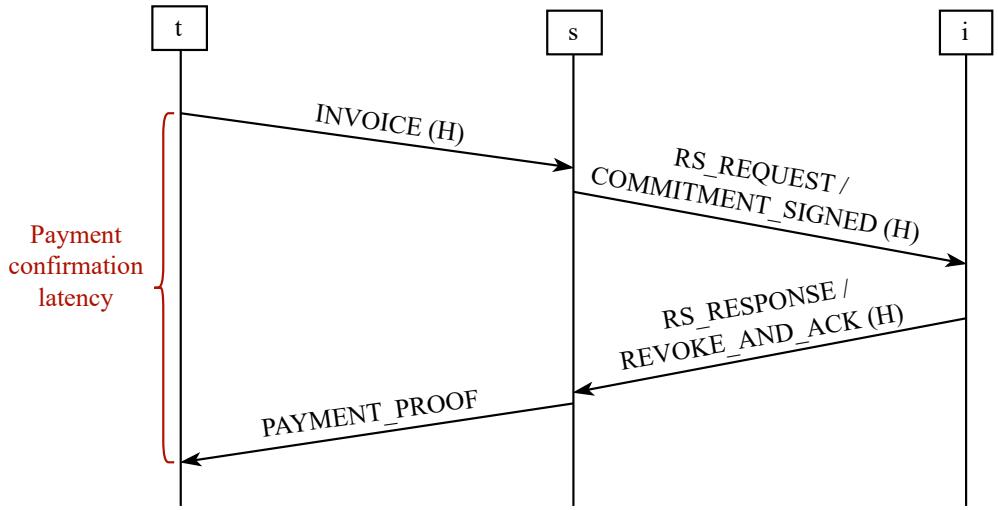

Figure 6.2: The messages involved in our proposed payment scheme. The payment recipient, t, receives an anticipated confirmation from the payment sender, s, before receiving the payment. The confirmation allows actions to be performed before the payment completes, reducing the application's latency. The messages involved in the scheme are piggy-backed into Lightning's peer-to-peer messages defined in BOLT#2 [3].

the sender s, λ_{MCF} is the time it takes to solve the minimum-cost flow problem and λ_{p_k} are the confirmation latencies for the payment parts as defined by Equation 6.5. This definition captures the fact that, in a regular payment operation, the confirmation latency of a multipath payment depends on the confirmation latency of the slowest payment part. Also, it shows that the capacity to solve the minimum-cost flow problem directly impacts payment confirmations.

6.3.1 Delayed Payments with Reduced Confirmation Latency

Having a confirmation latency that depends on the slowest path and a fast solution to the MCF problem can be prohibitive for time-sensitive applications, especially if the minimum-cost flow is to be computed by resource-constrained devices. Instead, we propose a payment scheme that anticipates payment confirmations while delegating path computations to a capable entry node. This scheme yields a *delayed payment* that is confirmed immediately but routed later. We depict the proposed confirmation scheme in Figure 6.2 and describe it below.

To begin a delayed payment after receiving an invoice from a destination t, the light node s sends an `RS_REQUEST` (RS_{req}) message to the entry node i containing the following fields:

$$RS_{req} = \langle Type | PK_t | H \rangle_{\sigma_s}, \tag{6.7}$$

where $Type = 0$ (request) is a message type identifier, PK_t is the public key of t,

which corresponds to its address in the PCN, H is the payment hash and σ_s denotes all the message fields are signed by s. The RS_{req} message signals a routing service request from s to i, indicating that i should compute paths and deliver the payment to t. The message is piggy-backed into a `COMMITMENT_SIGNED` message that contains a new commitment transaction signed by s. The commitment transaction, in turn, includes the first HTLC of the payment chain. In this case, we skip the `UPDATE_ADD_HTLC` message and place the HTLC directly into the `COMMITMENT_SIGNED` message to minimize latency.

Upon receiving the routing service request, the entry node i evaluates it and responds with a signed `RS_RESPONSE` (RS_{resp}) message containing the following fields:

$$RS_{resp} = \langle Type | Resp | RS_{req} \rangle_{\sigma_i}, \tag{6.8}$$

where $Type = 1$ (response) is the message type identifier, $Resp$ is i's response (0 if refused, 1 if accepted), RS_{req} is the routing service request to which the response refers to, and σ_i denotes i's signature over the message fields. If i accepts the request, it commits the received HTLC and piggy-backs the RS_{resp} message into a `REVOKE_AND_ACK` message to s, signaling the HTLC has been committed.

When the routing service response reaches s, it constructs a `PAYMENT_PROOF` (P_{proof}) message

$$P_{proof} = \langle Type | RS_{resp} | CommitTX \rangle_{\sigma_s}, \tag{6.9}$$

where $Type = 2$ (proof) is the message type identifier, RS_{resp} is the routing service response message to which the proof refers, and $CommitTX$ is the commitment transaction signed by i that committed the first HTLC of the payment into the channel state. Including the commitment transaction into the payment proof message ensures s cannot lie about the current status of the HTLC, with the downside of revealing the state of channel (s, i) to t. We decide to disclose the channel state because we consider t can obtain it using targeted probing techniques, so preserving channel state secrecy does not yield strong privacy guarantees [147, 148]. We plan to adopt lightweight zero-knowledge proofs in future versions of our scheme [157].

The payment is confirmed once t receives the payment proof. The reduced confirmation latency of a delayed payment in the proposed scheme can thus be defined as

$$\Lambda_{red}(f) = \lambda_{inv} + \lambda_{RS_{req}} + \lambda_{RS_{resp}} + \lambda_{P_{proof}}, \tag{6.10}$$

where λ_{inv}, $\lambda_{RS_{req}}$, $\lambda_{RS_{resp}}$, $\lambda_{P_{proof}}$ are the propagation delays of the invoice, routing service request, routing service response, and payment proof messages, respectively. Considering the messages do not transmit significant amounts of data, and that propagation delays are roughly symmetrical, we can approximate the delays using

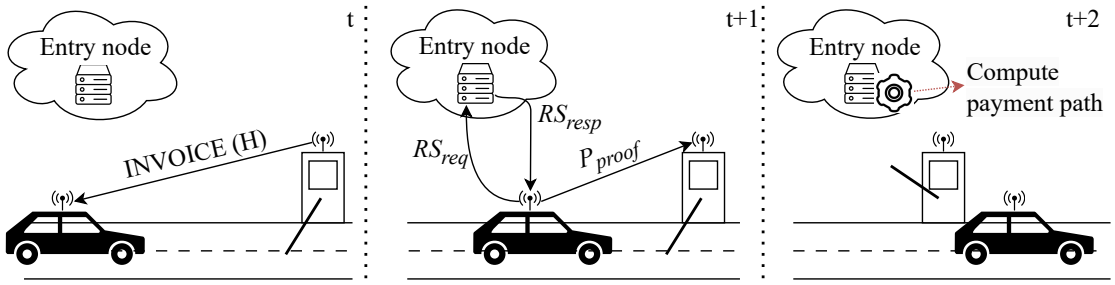

Figure 6.3: An example of a time-sensitive electronic toll collection application that adopts the proposed payment scheme. The application leverages the reduced confirmation latency to open the toll gate without stopping the car.

round-trip times, which yields

$$\Lambda_{red}(f) = RTT_{st} + RTT_{si}, \tag{6.11}$$

where RTT_{st} and RTT_{si} are, respectively, the round-trip times of the communication channel between s and t and s and i. Note that $\Lambda_{red}(f) \ll \Lambda(f)$ as $\Lambda_{red}(f)$ depends only on the propagation delays of two channels while $\Lambda(f)$ depends on the propagation delays of all channels in the payment's path.

We present a use case to illustrate how anticipated confirmations and delayed payments work in our proposed scheme in Figure 6.3. In the example, a car or a device in a car needs to connect to a wireless toll collector and pay the toll without stopping. With a default payment scheme, the first three phases of the regular payment process (invoice forwarding, pathfinding, and route locking) must finalize before the car leaves the communication range of the toll collector. The process yields a latency of $\Lambda(f)$, as modeled in Equation 6.6, which will likely stop the car until the payment is delivered. However, in our payment scheme, the gate can be opened quickly since the car confirms the payment as soon as it establishes the first HTLC with an entry node. If the car does not have an active connection when it traverses the toll system, we assume it can use the toll booth's antenna as an access point to communicate with the entry node.

Our proposed payment scheme requires two modifications to the default Lightning protocol described in the BOLTs [3, 76, 85]. First, the `COMMITMENT_SIGNED` message is modified to contain a previously-unseen HTLC. This modification eliminates the delays in sending `UPDATE_ADD_HTLC` messages and waiting for node timeouts at the expense of a larger `COMMITMENT_SIGNED` message. Second, the new routing service messages, RS_{req}, RS_{resp}, must be piggy-backed with the messages of Lightning's channel state update protocol, defined in BOLT#2 [3]. As the Lightning BOLTs explicitly support custom message types and lengths, both modifications can be implemented as an add-on feature for channels with resource-constrained devices on

Lightning implementations [4–6].

6.3.2 Payment Security and Adversary Model

Anticipating payment confirmation incurs a cost to payment security. While in a regular payment, the payment confirmation represents an irrevocable guarantee that the recipient can redeem its coins, an anticipated confirmation only proves that the payment sender initiated the payment. The delayed payment may still not reach the recipient due to route failures or unexpected behavior by the entry node that is supposed to route the payment. In particular, malicious entry nodes represent the main vulnerability in our scheme, as they can easily prevent the payment from being forwarded to the correct destination.

We assume all messages are signed, and the communication channels between light devices and entry nodes are secure so that no message can be eavesdropped. Regarding entry nodes, we consider an adversary model in which malicious entities can perform the following actions:

1. Attempt to steal coins from channels with light nodes by publishing revoked channel states;
2. Route payments to the wrong destination;
3. Accept a routing service request and not route the payment.

We mitigate the threats as mentioned earlier as follows. First, we assume payment recipients close their payment channels before disconnecting or adopt a coin-theft protection mechanism such as watchtowers [25, 158] or minimum lock-time windows [159]. This assumption prevents entry nodes from publishing revoked states without getting punished, as discussed in Section 4.3. Payment senders do not need protection mechanisms since publishing revoked states, in this case, incurs fund losses to the entry node. Second, we add a *routing budget* to payments as a financial incentive for entry nodes to route the payments honestly. Entry nodes gain the routing budget if and only if the payment is delivered to its correct destination, mitigating malicious behavior. We detail how these budgets work in Section 6.4.

An important assumption we make is that entry nodes *cannot* collude with payment senders. Otherwise, the sender can provide a payment proof issued by an adversary entry node who intentionally refuses to forward the payment afterward until the HTLC expires. Routing budgets do not mitigate this behavior because it is more financially advantageous for the entry node to split the entire payment's value with the sender after the HTLC times out than to gain the routing budget for honesty. Worse, if the same entity controls the entry node and the sender, that

entity can issue fake payments for free. Mitigating this threat systematically involves setting a reputation system for entry nodes, such that payment proofs are only accepted if they come from nodes with enough reputation. As reputation systems are currently under development in the Lightning Network [160], we believe a systematic reputation-based solution for entry nodes will soon appear.

For the purposes of this work, we assume entry nodes are selected from a list of nodes that the recipient trusts. The list does not need to be an allowlist; instead, payment recipients can adopt an optimistic approach in which proofs are refused if they come from a node previously identified as malicious. Adopting allowlists can be especially useful for small payments, which incur less financial losses if embezzled. However, the payment recipient should decide to adopt an allowlist or a denylist at their discretion, case by case. If the sender connects only to entry nodes the recipient considers malicious, it can compute partial paths and use trampoline routing [151] to send the payment to a trusted node securely. This involves computing a partial path over a small graph, which could be better but is feasible given the exceptional circumstances.

6.4 Finding Constrained Optimal Flows

The second part of the routing problem with payments from resource-constrained devices involves effectively routing a payment after it is relayed to some entry node. This problem is similar to the minimum-cost flow problem enunciated in Definition 9, except it considers metrics that reflect the specific requirements of applications with resource-constrained devices. This work considers three metrics: routing budgets, HTLC-resolution delays, and delivery probabilities.

Routing budgets. Routing budgets represent an extra cost that resource-constrained senders must pay in return for the routing service provided by some entry node. The budget incentivizes entry nodes to provide correct routes to delayed payments, reducing the risk of payment losses. We name them "budgets" because they define an upper bound to the routing fees that can be paid when routing a payment:

$$R_B \geq \sum_{\pi_k \in f} \sum_{(i,j) \in \pi_k} \phi_{ij} f_{ij}, \tag{6.12}$$

where f is the flow composed of all payment paths π_k, f_{ij} is the amount of flow in arc (i, j), and ϕ_{ij} is the arc's routing fee. When given a routing budget R_B, the goal of an entry node is to maximize its profit, expressed by:

$$\Gamma = R_B - \sum_{\pi_k \in f} \sum_{(i,j) \in \pi_k} \phi_{ij} f_{ij}, \tag{6.13}$$

or, in words, entry nodes receive the difference between the given routing budget and the fees charged for routing the payment. Consequently, rational entry nodes will attempt to find a set of feasible paths that minimizes the total cost of routing fees. This behavior yields a version of the minimum-cost flow problem presented in Equation 6.4 in which the arc costs are routing fees:

$$\min_{f_{ij} \in \mathbb{N}_0} \sum_{\pi_k \in f} \sum_{(i,j) \in \pi_k} \phi_{ij} f_{ij}. \tag{6.14}$$

Payment senders incorporate routing budgets into the value of the first HTLC in the payment path such that $HTLC_V = R_B + P_V$, where P_V is the value of the payment (the amount that reaches the recipient after all intermediaries subtract routing fees). Payment recipients should set the price of routing budgets as they are the entity at risk in a delayed payment. R_B values must be high enough to make the payment feasible and profitable for entry nodes. On the other hand, values of R_B that are too high may scare customers since they incur extra fees that are not part of the product's value. Hence, the choice of R_B represents a trade-off between payment security and product attractiveness: if R_B is too high, the payment is likely to be delivered, but the customer may give up paying for the product at all; if R_B is too low, the product becomes cheap, but the risk of payment loss is high. In practice, the choice of R_B depends entirely on the structure of the network, the payment value, the product, and the risk the recipient is willing to take. After selecting a value for R_B, recipients pass the desired routing budget along the invoice message to senders.

HTLC-resolution delays. Apart from routing budgets, payments from light nodes must also set HTLC timeouts that match the connectivity patterns of resource-constrained devices. Else, the entry node may decide to close the channel with the payment source in case the source temporarily disconnects. We clarify this behavior with an example below.

Let s be a resource-constrained device that sends a payment to a destination t and $HTLC_{ij}(V, T)$ denote an HTLC from i to j with value V and timeout T. Assume s establishes $HTLC_{si}(R_B + P_V, T)$ with the entry node i using the delayed payment scheme and disconnects. Also, assume the payment is single-path for simplicity. After s establishes the first HTLC, the entry node routes the payment to t, and then t triggers the unlocking phase, in which each intermediary claims the HTLCs in the path until the process eventually reaches channel (s, i). At this point, i cannot claim $HTLC_{si}$ off-chain because s is disconnected and consequently incapable of updating the channel state. Thus, i must decide: wait for s to reconnect or close the payment

channel and claim its funds on-chain. In the best-case scenario, i decides to wait, s reconnects quickly, and the payment finishes before it times out. In the worst case, i claims the HTLC on-chain before the timeout T expires; otherwise, s can cancel the HTLC when it reconnects. Recall that claiming an HTLC on-chain involves publishing the channel-closing transaction, which causes i to pay transaction fees and wait a long period until the transaction appears in the blockchain.

As T is defined by s, the goal of i is to maximize this "grace period" in which it waits for s to reconnect, avoiding channel closures. If we ignore block validation times for simplicity, the grace period can is

$$G_P = T - \sum_{(i,j)\in\pi_k} \Delta_{ij}, \tag{6.15}$$

where Δ_{ij} are the maximum HTLC-resolution delays accepted by each arc (i, j) (recall from Section 3.3.1 that nodes enforce a minimum timeout difference per arc so they have enough time to claim HTLCs after receiving the payment preimage). Thus, the sum of Δ_{ij}'s should be as low as possible to enlarge the grace period and reduce the chance of forced channel closures. Maximizing the grace period for all paths in a multipath payment yields a *fixed-cost* MCF problem in which the arc costs are the HTLC-resolution delays Δ_{ij}:

$$\min_{f_{ij}\in\{0,1\}} \sum_{\pi_k\in f} \sum_{(i,j)\in\pi_k} \Delta_{ij} f_{ij}. \tag{6.16}$$

Like routing budgets, the timeout T of the first HTLC represents an upper bound on the time it takes to finalize a payment *in the worst case*. Note, however, that the value of the grace period does not impact payment latency when all nodes in the path are responsive.

Delivery probabilities. The last challenge of payment routing with resource-constrained devices is dealing with packet losses and the uncertain channel liquidities presented in Section 6.1.2. These two elements cause HTLCs to be refused or lost, which in turn causes payment failures. The possibility of failures introduces a per-arc probability metric that quantifies how likely an HTLC is to be established in the arc or, equivalently, how likely a payment is to traverse the arc. The *success probability* of an arc (i, j) is

$$p_{ij} = P(f_{ij} \leq l_{ij}) \times (1 - P(\epsilon_{ij})), \tag{6.17}$$

where $P(f_{ij} \leq l_{ij})$ is the probability that the arc has enough liquidity to forward f_{ij} coins and $P(\epsilon_{ij})$ represents the probability of HTLC errors due to packet losses or unresponsive nodes. Note that $p_{ij} = P(f_{ij} \leq l_{ij})$ if the underlying communication

channel is fully reliable and nodes are always online. Using p_{ij}, we can compute the probability that a single-path payment part k is delivered as

$$p_k = \prod_{(i,j)\in\pi_k} p_{ij}, \tag{6.18}$$

where π_k is the payment's path. Delivery probabilities directly impact the efficiency of payment routing algorithms, as payment parts that fail must be retried or abandoned. Thus, another goal for routing nodes is to maximize the probability of payment deliveries so that payments are complete in the least amount of attempts. To make the problem linear, we can use the negative natural logarithm of probabilities p_{ij}, such that we obtain another fixed-cost MCF problem for multipath payments:

$$\min_{f_{ij}\in\{0,1\}} \sum_{\pi_k\in f} \sum_{(i,j)\in\pi_k} -ln(p_{ij})f_{ij}. \tag{6.19}$$

We highlight that negative logarithms transform delivery probabilities into an additive metric that can be computed per path using shortest path algorithms [144, 145].

Combining metrics into a single MCF problem. As we hinted in the previous sections, routing payments implies minimizing a metric, which can be routing fees, HTLC-resolution delays, delivery probabilities, or any other metric that might be important for an application. However, we generally cannot provide a single optimal solution concerning all metrics, as such a solution is likely to not exist. Instead, we can: (i) combine the multiple independent metrics into a single mixed-metric (SMM) such that this new metric can be minimized or (ii) select a preferred metric to minimize and set constraints to the secondary metrics such that we prune solutions that violate predefined budgets.

Although several payment routing proposals adopt SMM approaches [133, 134, 161], we argue that choosing a main metric while constraining the others is the best option for PCNs. First, almost all routing proposals consider routing fees as the only metric in the problem, which demonstrates a clear bias towards this metric [100, 103, 126, 132, 136–140]. This preference makes perfect sense since minimizing routing fees directly impacts how much money users spend to send payments. Second, optimizing one of the other metrics mentioned in this work does not give an equally important advantage to payments in general. For instance, minimizing HTLC-resolution delays is only critical when nodes are expected to disconnect for long periods. Maximizing payment delivery probabilities is mostly fit for applications that need to avoid payment retries. Although we recognize that these metrics might be essential in some use cases, we usually want to minimize routing fees subject to minimal levels of grace periods and delivery probabilities for

payments with resource-constrained devices. In other words, we want "the cheapest path that will likely allow us to deliver the payment and prevent channel closures in case the payment source disconnects". Lastly, SMM-based approaches raise the problem of combining metrics in different scales, such as routing fees ($\phi_{ij} \in [0, \infty]$) and delivery probabilities ($p_{ij} \in [0, 1]$), and the problem of how to weight each metric in the metric-mixing equation [134].

Choosing a main metric allows us to reformulate the MCF problem presented in Definition 9 as a *constrained* problem in which the side constraints set bounds to the secondary metrics:

Definition 10. *(Constrained minimum-cost flow problem).* Given a transportation network $G_l = (V, A)$ with liquidities l_{ij} $\forall (i,j) \in A$, a metric set $\mathcal{W} = \{w_1, ..., w_n\}$, and a demand d to be transferred from a source node s to a sink node t, the constrained minimum cost-flow problem is an optimization problem in the form:

$$\min_{f_{ij} \in \mathbb{N}_0} \quad \sum_{(i,j) \in A} c_{ij}^{\sigma} f_{ij}, \tag{6.20a}$$

$$\text{subject to:} \quad 0 \leq f_{ij} \leq l_{ij} \qquad \forall (i,j) \in A, \tag{6.20b}$$

$$\sum_{(i,j) \in A} f_{ij} - \sum_{(j,i) \in A} f_{ji} = \begin{cases} d, \text{if } i = s \\ -d, \text{if } i = t \\ 0, \text{otherwise} \end{cases} \qquad \forall i \in V, \tag{6.20c}$$

$$\sum_{(i,j) \in A} c_{ij}^{w} \mathbb{1}_{f_{ij} > 0} \leq B^{w} \qquad \forall w \in \mathcal{W} \setminus \sigma, \tag{6.20d}$$

where c_{ij}^{σ} is the cost of arc (i,j) w.r.t. the main metric $\sigma \in \mathcal{W}$, c_{ij}^{w} are the costs of arc (i,j) w.r.t. the secondary metrics $w \in \mathcal{W} \setminus \sigma$, B^{w} are the upper bounds of each secondary metric, and $\mathbb{1}_{f_{ij}>0}$ is an indicator function that outputs 1 if a non-zero flow traverses arc (i,j), i.e., if $f_{ij} > 0$.

A constrained minimum-flow (CMCF) problem differs from the MCF formulation presented in Definition 9 in two aspects. First, the arc costs c_{ij} are replaced by costs c_{ij}^{σ} to denote that they refer to a user-set main metric σ. Second, the problem includes one extra constraint per secondary metric w. The extra constraints ensure the optimal solution w.r.t. the main metric does not violate predefined upper bounds on any secondary metric. Note that the costs c_{ij}^{w} form a per-arc cost vector $\vec{c}_{ij} \in \mathbb{R}^n$, and that we minimize the cost of the main metric in this vector while constraining the other costs. In contrast, an SMM approach would combine all costs into a single cost c_{ij} using a convex cost function and solve the MCF problem normally.

Similar to flows in the MCF problem, we can dissect constrained flows into constrained paths. Each constrained path π has an associated cost vector $\vec{c}_{\pi} \in \mathbb{R}^n$

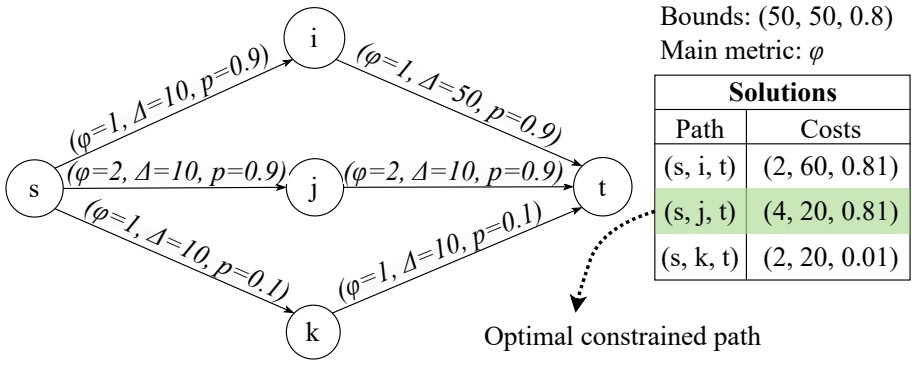

Figure 6.4: An example of constrained shortest path problem with three metrics: routing fees (ϕ), HTLC-resolution delays (Δ), and delivery probabilities (p). For clarity, we show the original probabilities instead of their negative logarithms, yielding a lower bound instead of an upper bound. The optimal solution is the path that minimizes the main metric while not violating any bounds.

expressed by

$$\vec{c}_\pi = \left(\sum_{(i,j)\in\pi} c_{ij}^{w_1}, ..., \sum_{(i,j)\in\pi} c_{ij}^{w_n} \right), \tag{6.21}$$

and constrained by the bound vector $\vec{B} = (B^{w_1}, ..., B^{w_n})$ such that $c_\pi^w \leq B^w \ \ \forall w \in \mathcal{W}$.

To find the constrained minimum-cost flow, we can push as much flow as possible through the constrained paths with minimum cost w.r.t. the main metric σ. Finding minimal-cost paths subject to side constraints is known as the constrained shortest path (CSP) in the literature [162, 163]. We show an example of such a problem with multidimensional arc costs in Figure 6.4. In the example, we want to minimize routing fees subject to time differences and delivery probabilities constraints. Note that, despite providing fewer routing fees, paths (s, i, t) and (s, k, t) violate the bounds for the secondary metrics and are not selected.

This work proposes two routing algorithms that route payments optimally through constrained shortest paths. The first algorithm is Generalized Pulse (GenPulse) which provides constrained shortest paths for single-path payments considering an arbitrary number of metrics. Next, we propose Multipath Pulse (MultiPulse), a multipath algorithm that solves the CMCF problem through sequential flow allocations along the constrained shortest paths provided by GenPulse. MultiPulse is inspired by the classical Successive Shortest Path algorithm for MCF problems [164, 165]. GenPulse and MultiPulse are named after the Pulse algorithm [162], a state-of-the-art method for solving CSP problems with one side constraint. We describe GenPulse and MultiPulse in the following sections.

6.4.1 The Generalized Pulse Algorithm

The Generalized Pulse (GenPulse) algorithm is a pathfinding algorithm that solves the constrained shortest path problem in a generic graph with multiple constraints. The algorithm is based on pulses propagating through a network from a source s to a destination t. Pulses are like particles that start at s and traverse the network from node to node, building partial paths π' with the visited nodes. Each partial path has a cost vector $\vec{c}_{\pi'}$ as defined in Equation 6.21. When a pulse reaches the destination t, it contains a feasible constrained path π from s to t. The GenPulse algorithm represents a generalization of the Pulse algorithm [162] for an arbitrary number of metrics.

When pulses propagate aimlessly, GenPulse enumerates all possible paths from s to t, ensuring that the optimal path π^* is inside the candidate path set. However, as enumerating all paths is prohibitively costly, the algorithm adopts pruning strategies to reduce the solution set so that it finds the optimal path π^* in a reasonable time. This idea is similar to traditional branch-and-bound algorithms, which perform solution enumeration and then prune sub-optimal candidate solutions. Pulses in GenPulse are checked at execution time, saving computational effort whenever a candidate path is pruned. Consequently, the strength of GenPulse lies in the efficiency of its pruning strategies.

Let $G = (V, A)$ be a directed graph with arc costs $\vec{c}_{ij} \in \mathbb{R}^n \ \forall (i,j) \in A$ and $\vec{B} \in \mathbb{R}^n$ be a bound vector containing the upper bounds for each metric in the problem. We can prune pulses that propagate through G using two criteria: bound violations or path dominance.

Pruning by bound violations. The idea of bound-based pruning is to interrupt pulses as soon as we observe they cannot reach the destination without violating some side constraint. To accomplish this, the algorithm must first compute the minimum costs needed to go from any node i in the graph to the destination t w.r.t. every metric w. During initialization, we reverse the arcs in G, creating a reversed graph $G_{rev} = (V, A')$ where $A' = \{(j,i) | (i,j) \in A\}$ and $c^w_{ji} = c^w_{ij}$. Next, the algorithm runs n one-to-all shortest path algorithms to find the distances $dist(t, i, w)$ from t to every node i w.r.t. metric w in the reverse graph. The reverse distances represent the minimum costs needed to reach t from any node i in the original graph G[7]. Then, using the pre-computed reverse distances, the algorithm can check if a partial path $\pi' = \{(s,u), ..., (v,i)\}$ yields a cost $c^w_{\pi'} > B^w - dist(t,i,w)$ for any metric w. If the check for a metric w is positive, pulses propagating from i will violate constraint B^w upon reaching t. Thus, the algorithm can prune such pulses. Furthermore, we can

[7]We could also obtain the minimum costs by running a shortest path algorithm from every node to the destination t. However, this approach is way slower in practice.

Algorithm 1: The Generalized Pulse (GenPulse) algorithm.

Input: $G, s, t, \vec{B}, \sigma$
Output: Optimal constrained path π^*_σ and its cost vector $\vec{c}_{\pi^*_\sigma}$

```
/* Generate graph with reversed edges                                    */
G_rev ← reverse(G)
/* Pre-compute distance matrix D containing the distances between t and
   all nodes in the reverse graph for each metric                        */
D ∈ ℝ^{n×V} ← 0
for w ∈ 𝒲 do
    for i ∈ G_rev do
        D_wi ← dist(t, i, w)
/* Sort neighborhoods by the main metric σ                               */
for i ∈ G do
    𝒮_σ(i) ← sort(𝒩(i), σ)
/* Initialize path and cost vector                                       */
π_0 ← ∅
c⃗_{π_0} ∈ ℝ^n ← 0
/* Call GenPulse routine with initial parameters                         */
GenPulseRoutine(s, π_0, c⃗_{π_0}, B⃗)
return π*_σ, c⃗_{π*_σ}
```

update the bound B^σ for the main metric every time a pulse reaches t, ensuring we prune worse paths than the current best solution.

Pruning by path dominance. We can define dominance relationships between partial paths as a way to prune pulses that we know are sub-optimal. Let π_A and π_B be two partial paths to a node i. If $c^w_{\pi_A} < c^w_{\pi_B}\ \forall w \in \mathcal{W}$, then we say π_A strongly dominates π_B since it has lower costs w.r.t. every metric. Similarly, if $c^w_{\pi_A} < c^w_{\pi_B}$ for some metric w and $c^x_{\pi_A} \leq c^x_{\pi_B}$ for any metric x other than w, we say π_A weakly dominates π_B. Given these relations, we can store the cost vectors of the most dominant partial paths that traverse each node i. By default, GenPulse stores the cost vectors of the best paths w.r.t. each metric w, as this yields the most efficient pruning results in the original Pulse paper [162]. Then, we compare the costs of new partial paths against the stored costs and prune the new path if it is dominated by a partial path already traversed i. Note that the optimal solution is always dominant as it provides the minimum cost w.r.t. the main metric.

We provide GenPulse's pseudo-code in Algorithm 1 and Algorithm 2. First, the algorithm reverses the graph G and computes the reverse distance matrix $D \in \mathbb{R}^{n \times V}$ using Dijkstra's algorithm with the metrics as arc weights. Then, it sorts the neighbors of each node i by the main metric σ so we explore the nearest neighbors first during the algorithm's execution. Next, we initialize the partial path π_0 as an empty set and the costs $\vec{c}_{\pi_0}$ as an all-zero vector and call GenPulse's recursive routine.

Algorithm 2: GenPulse routine.

Input: $i, \pi, \vec{c}_\pi, \vec{B}$
Output: void

if i = t **then**
  /* If we reached the target t, we found a new best (feasible) path. Save it and update the current bound for the main metric σ. */
  $\pi^*_\sigma \leftarrow \pi$
  $\vec{c}_{\pi^*_\sigma} \leftarrow \vec{c}_\pi$
  $B^\sigma \leftarrow c^\sigma_\pi$
  return
/* Obtain a new cost vector for every neighbor j of i (sorted by σ) */
for $j \in \mathcal{S}_\sigma(i)$ **do**
  $\vec{c}_{\pi'} \leftarrow \vec{c}_\pi + \vec{c}_{ij}$
  /* Check if any bounds will be violated and check if we already know a path that dominates this */
  if `checkBounds`$(j, \vec{c}_{\pi'}, D, \vec{B})$ and `checkDominance`$(j, \vec{c}_{\pi'})$ **then**
    $\pi' \leftarrow \pi \cup (i, j)$
    `GenPulseRoutine`$(j, \pi', \vec{c}_{\pi'}, \vec{B})$

GenPulse's routine propagates pulses along nodes and prunes them according to bound violations and path dominances. When the algorithm finishes, it returns the shortest constrained path w.r.t. the main metric, π^*_σ, and its cost vector, $\vec{c}_{\pi^*_\sigma}$.

Finally, we highlight that GenPulse presents a few differences compared to the original Pulse algorithm [162]. First, Pulse only supports single-metric graphs with a single side constraint, while GenPulse supports multi-metric graphs with unlimited side constraints. Second, GenPulse's bound-pruning strategy unifies Pulse's bound and feasibility-pruning strategies into a single strategy. Lastly, we implement GenPulse in Python language[8], while Pulse is implemented in Java[9].

6.4.2 The Multipath Pulse Algorithm

Multipath Pulse (MultiPulse) is a multipath routing algorithm that leverages GenPulse's constrained shortest paths to find constrained minimum-cost flows. The algorithm is inspired by the classical Successive Shortest Paths (SSP) algorithm [164], a method for solving minimum-cost flow problems based on consecutive flow pushes along the k-shortest paths between two nodes in a graph. The SSP algorithm provides an exact solution to the MCF problem when arc capacities are known [165]. Like SSP, MultiPulse relies on the concept of residual networks to push flows:

[8]Available at https://github.com/gfrebello/wpcn-routing/blob/main/multipath/genpulse.py
[9]Available at https://github.com/dukduque/jPulseBase

Definition 11. *(Residual network).* Given a capacitated transportation network $G = (V, A)$, replace each arc $(i, j) \in A$ by two arcs (i, j) and (j, i). The arc (i, j) has cost c_{ij} and residual capacity $r_{ij} = l_{ij} - f_{ij}$, where l_{ij} is the arc's original capacity and f_{ij} is the flow on (i, j). The arc (j, i) has cost $c_{ji} = -c_{ij}$ and residual capacity $r_{ji} = f_{ij}$. Remove arcs with zero residual capacity. The resulting network is a residual network $R_f = (V, A')$, which represents the result of pushing a flow f through G.

The MultiPulse algorithm works by iteratively augmenting a flow through feasible paths until it obtains a maximum flow. This common strategy leverages the well-known max-flow min-cut theorem [166] to obtain maximum flows by removing arcs in a residual network. However, because we aim for the constrained minimum-cost flow instead of any maximum flow, we augment the flow along the constrained shortest paths instead of those with the largest capacities. Furthermore, we limit the maximum flow by introducing additional arcs that act as a bottleneck for the problem.

Let $G = (V, A)$ be a directed graph with arc costs $\vec{c}_{ij} \in \mathbb{R}^n \ \forall (i, j) \in A$ and capacities $l_{ij} \ \forall (i, j) \in A$. Let d be a demand to be transported from source s to sink t. The actions MultiPulse executes to find a constrained minimum-cost flow are:

1. Initialize the flow f to a zero-amount flow;
2. Initialize the residual network to $R_f = G$ since no flow is pushed;
3. Add two arcs, (s', s) and (t, t'), to R_f with all-zero cost vectors $\vec{c}_{s's} = \vec{c}_{tt'} = (0, ..., 0) \in \mathbb{R}^n$ and residual capacities $r_{s's} = r_{tt'} = d$;
4. Find the constrained shortest path π^*_σ from s' to t' w.r.t. the main metric σ in R_f using the GenPulse algorithm. Obtain the corresponding cost vector $\vec{c}_{\pi^*_\sigma}$;
5. Push as much flow as possible along π^*_σ. The maximum flow that can be pushed is $f(\pi^*_\sigma) = \min(l_{ij} \ \forall (i, j) \in \pi^*_\sigma)$. Save π^*_σ, $\vec{c}_{\pi^*_\sigma}$, and $f(\pi^*_\sigma)$;
6. Update the residual network R_f as described in Definition 11;
7. Repeat steps 4, 5, and 6 until there is no path from s' to t';

We provide the pseudo-code of MultiPulse in Algorithm 3. In each round of the algorithm, we obtain the path-cost-amount 3-tuple $PCA_k = (\pi^*_\sigma, \vec{c}_{\pi^*_\sigma}, f(\pi^*_\sigma))$, where π^*_σ is the shortest path between s' and t' w.r.t. σ, $\vec{c}_{\pi^*_\sigma}$ is the cost vector of π^*_σ, and $f(\pi^*_\sigma)$ is the amount of flow the algorithm successfully pushed through π^*_σ in the k-th iteration. We represent thus the optimal flow f^* as a set $\mathcal{PCA} = \{PCA_k \mid k \in \{1, ..., K\}\}$ containing all the 3-tuples found during the algorithm's execution. The MultiPulse

algorithm is guaranteed to converge to the optimal solution for the constrained minimum cost flow problem. We provide a formal proof for this statement in Appendix A.4.

Algorithm 3: The Multipath Pulse (MultiPulse) algorithm.

Input: $G, s, t, d, \vec{B}, \sigma$
Output: Path-cost-amount 3-tuple set $\mathcal{PCA}$

/* Initialize empty 3-tuple set */
$\mathcal{PCA} \leftarrow \emptyset$
/* Initialize residual graph R */
$R \leftarrow G$
/* Add arcs to limit the maximum amount to the demand d */
Add arc (s', s) with capacity $l_{s's} = d$ and cost vector $\vec{c}_{s's} = 0$ to R
Add arc (t, t') with capacity $l_{tt'} = d$ and cost vector $\vec{c}_{tt'} = 0$ to R
while $\exists$ a path from s' to t' in R **do**
    /* Obtain shortest constrained path through GenPulse */
    $\pi^*_\sigma, \vec{c}_{\pi^*_\sigma} \leftarrow \texttt{GenPulseAlgorithm}(R, s', t', \vec{B}, \sigma)$
    /* Augment flow along π^*_σ. PushFlow returns the amount of flow pushed through π^*_σ, $f(\pi^*_\sigma)$, and the resulting residual graph */
    $R, f(\pi^*_\sigma) \leftarrow \texttt{PushFlow}(R, \pi^*_\sigma, \sigma)$
    /* Update 3-tuple set */
    $\mathcal{PCA} \leftarrow \mathcal{PCA} \cup (\pi^*_\sigma, \vec{c}_{\pi^*_\sigma}, f(\pi^*_\sigma))$
return $\mathcal{PCA}$

6.5 Proof-of-Concept Evaluation

We implement a prototype of GenPulse and MultiPulse in Python language[10] using the `networkx` library[11]. Our prototypes consider a CMCF problem with three metrics: the routing fees, HTLC-resolution delays, and delivery probabilities discussed in previous sections. We adopt routing fees as the main metric.

To facilitate result visualization, we introduce an auxiliary variable ψ that quantifies the tightness of side constraints in the CSP (recall that the efficiency of GenPulse depends on bound-based pruning strategies). Given a set of metrics $w \in \mathcal{W}$, we define the tightness of each constraint as

$$\psi_w = 1 - \frac{B^w - dist(s, t, w)}{B^w}, \tag{6.22}$$

where $dist(s, t, w)$ is the cost of the shortest path between s and t w.r.t. metric w and B^w is the upper bound of w. A high value of ψ_w indicates the constraint for

[10] Available at https://github.com/gfrebello/wpcn-routing
[11] Available at https://networkx.org/

Table 6.2: Summary of the networks we use in our experiments.

Parameter	Scale-free PCN	LN snapshot
n	$\{512, 1024, 2048, 4096, 8192\}$	$\{512, 1024, 2048, 4096, 8192\}$ (snowball-sampled)
ϕ_{ij}	$U\{0, 100\}$	From dataset (see Fig. 6.5).
Δ_{ij}	$U\{50, 1000\}$	From dataset (see Fig. 6.5).
p_{ij}	$U[0.8, 1]$	$U[0.8, 1]$
l_{ij}	$U\{0, 10^6\}$	From dataset (see Fig. 6.5 and description in the text).

metric w is tight, whereas a low value indicates the constraint is loose. ψ_w yields a value in the interval $[0, 1]$ regardless of w's domain, which allows us to compute an overall tightness measure for the problem instance as

$$\psi = \frac{1}{n} \sum_{w \in \mathcal{W} \setminus \sigma} \psi_w, \tag{6.23}$$

where σ is the main metric and n is the number of metrics. Assuming, for simplicity, that the bounds for all secondary metrics are equally tight, we can derive bound vectors for GenPulse from ψ values as

$$\vec{B} = \frac{1}{\psi} \Big(dist(s, t, w_1), ..., dist(s, t, w_n) \Big) \in \mathbb{R}^n. \tag{6.24}$$

Hence, in the experiments, we obtain a bound vector $\vec{B}$ for each displayed value of ψ and use it to model constraints. Similar to ψ_w, high values of ψ indicate the bound vector is tight, whereas low values of ψ indicate the bound vector is loose.

6.5.1 Evaluation Setup

We analyze the performance of GenPulse and MultiPulse in a 24-core Intel Xeon Silver 4310 CPU server with 32 GB of RAM. Unless explicitly stated otherwise, we repeat each experiment 30 times to account for statistical variations, and each error bar indicates a 95% confidence interval. We run experiments in the two types of payment channel networks described below. Table 6.2 summarizes the parameters we use in each network.

Scale-free PCN. The topology of the first PCN is generated randomly using a scale-free network model. We choose the scale-free model because it is the random model that best captures the structure of the Lightning Network [87, 124]. Furthermore, we set the power-law exponent of the model to $\gamma = 2.1387$ as this is the value that yields the maximum-likelihood fitting of empirical Lightning data [87]. We sample the values of the routing fees ϕ_{ij}, HTLC-resolution delays Δ_{ij}, delivery probabilities p_{ij}, and arc liquidities l_{ij} from uniform (U) distributions, such that $\phi_{ij} \in \{0, ..., 100\}$, $\Delta_{ij} \in \{50, ..., 1000\}$, $p_{ij} \in [0.8, 1]$, and $l_{ij} \in \{0, ..., 10^6\}$. The intervals were defined

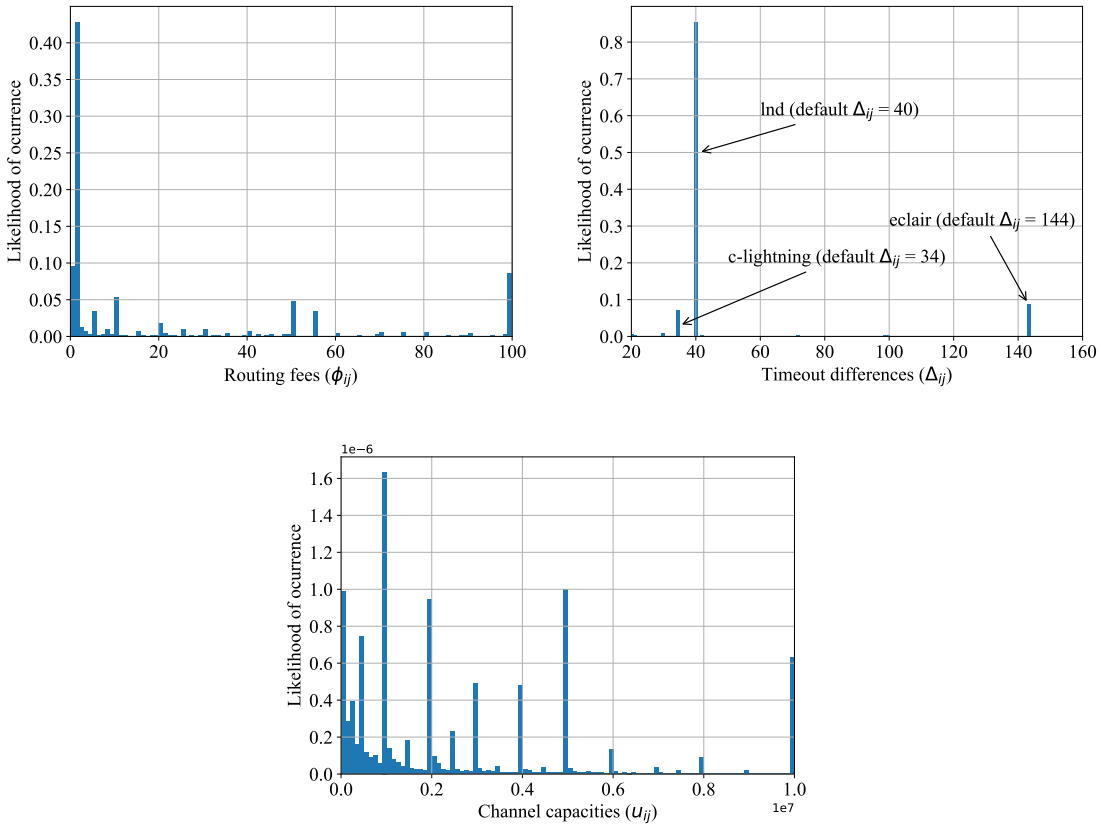

Figure 6.5: Distributions of routing fees (ϕ_{ij}), HTLC-resolution delays (Δ_{ij}), and channel capacities (u_{ij}) in the Lightning Network snapshot. The peaks in Δ_{ij} correspond to the default values of the main Lightning Network implementations [4–6].

to reflect realistic values in PCNs. We build 5 scale-free topologies with sizes $n \in \{512, 1024, 2048, 4096, 8192\}$.

Lightning Network (LN) snapshot. The second PCN is a snapshot of the Lightning Network built using a dataset of gossip messages[12] collected by the developers of the Core Lightning implementation [5, 81]. All messages in the dataset have a timestamp, which allows us to replay them in order and reconstruct the network topology at any given time. We reconstruct the topology as it was on the 1st of July 2022 and recover missing arc information using LN explorers, which yields a snapshot of approximately 14,000 nodes and 80,000 edges[13]. The routing fees ϕ_{ij}, the HTLC-resolution delays Δ_{ij}, and the channel capacities u_{ij} are obtained directly from the snapshot, as these are advertised in gossip messages. We show their distributions in Figure 6.5. The delivery probabilities are modeled like in the scale-free PCN, i.e., $p_{ij} \in [0.8, 1]$. For arc liquidities, we adopt two approaches: if the channel connects

[12]Available at https://github.com/lnresearch/topology.

[13]We reconstructed LN topologies for other moments using the same method. We make all reconstructed topologies available at https://gta.ufrj.br/ gabriel/files/ln-graphs.tar.gz.

two central nodes[14], we set $l_{ij} = \lfloor \frac{u_{ij}}{2} \rfloor$, and set the liquidity of the opposite arc to $l_{ji} = u_{ij} - l_{ij}$. Otherwise, we assume channel liquidities are heavily biased towards some node. We flip a coin to decide which node was the channel initiator i and set the liquidities to $l_{ij} = 0.99 \times u_{ij}$ and $l_{ji} = u_{ij} - l_{ij}$. We adopt this approach to reflect that central-central channels tend to be balanced while central-peripherical and peripheral-peripheral channels tend to be highly imbalanced [159][15]. Then, we simulate 50,000 payments between random nodes using PCNSim and the credit card payment dataset presented in Section 5.3.1 to recreate a realistic liquidity distribution. Finally, we snowball-sample [167] the graph from the most central node to create snapshots of sizes $n \in \{512, 1024, 2048, 4096, 8192\}$ for comparisons with the scale-free graphs.

6.5.2 GenPulse's Performance

First, we aim to analyze how long it takes for GenPulse to find a constrained shortest path in a network if such a path exists. To accomplish this, we select 50 random (s, t) pairs from the scale-free PCN and the Lightning Network snapshot and compute the average execution time of GenPulse for each graph. Note that among the (s, t) pairs, pairs can be close to each other so that pulses arrive quickly regardless of how tight the bounds are and pairs that are far away, such that many pulses propagate aimlessly unless we have tight bounds. By selecting 50 random pairs, we expect both effects will be covered, yielding a result that reflects the expected performance of GenPulse for some random (s, t) in the graph. We run the experiment for $\psi \in \{0.05, 0.1, ..., 1\}$ and all graph sizes $n \in \{512, 1024, 2048, 4096, 8192\}$.

Figure 6.6 depicts the performance results for GenPulse on both the scale-free PCN and the Lightning Network snapshot. We observe three main findings. First, increases in the network size cause a proportional increase in GenPulse's execution time for all ψ values. This increase likely occurs because larger networks imply more paths exist between s and t, which causes more pulses to propagate [80]. Similarly, an increase in the network size causes the CSP problem to be unsolvable for high values of ψ. This phenomenon occurs because the shortest paths tend to be longer in large networks, yielding higher costs that are more likely to violate some bound. Finally, for both networks, we observe that the tightness of the bound vector significantly impacts GenPulse's performance. This result confirms the assumption that the pruning strategies represent the main strength of GenPulse: the tighter the bounds, the more efficiently GenPulse directs pulses toward the constrained shortest

[14]Central nodes are nodes with high out-degree. We treat any node with degree $deg^+ > 20$ as a central node.

[15]This happens because most channels are unilaterally funded in LN [4–6]. In central-central channels, however, this imbalance is corrected with rebalancing techniques to ensure the channel can route transactions efficiently.

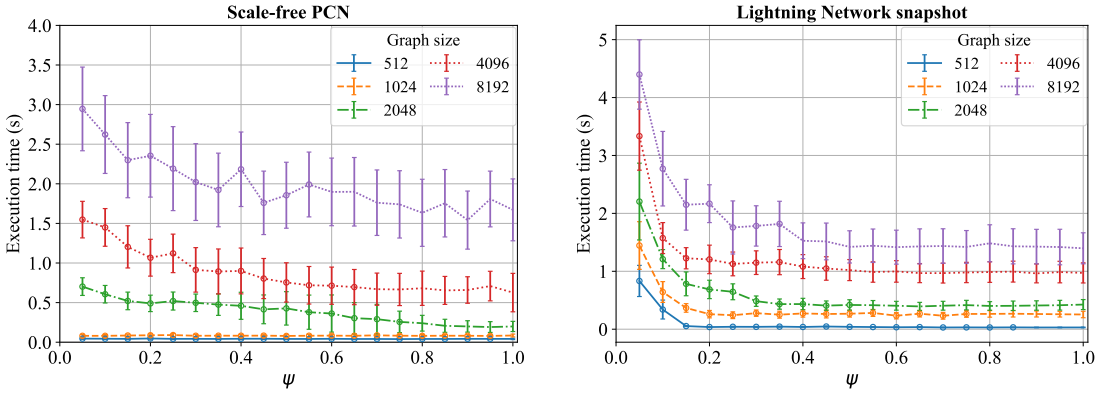

Figure 6.6: Performance of the GenPulse algorithm for several graph sizes and ψ values in a randomly-generated scale-free network and a Lightning Network snapshot. The markers indicate a constrained shortest path has been found. The results demonstrate that the tightness of side constraints significantly impacts the algorithm's performance.

path. The result is consistent with Pulse's original results for a single constraint constraint [162]. Thus, GenPulse should be used primarily to find payment paths in use cases in which the entry node needs tight constraints, e.g., when nodes disconnect for long periods and when the probability of HTLC loss is high.

A result not shown in the figure is that some rare executions of the algorithm for the Lightning Network take an outstanding amount of time to finish when ψ is very low. Besides the obvious reason that a low ψ causes high execution times, we believe this behavior occurs specifically in the Lightning Network snapshot because the distribution of routing fees is not uniform. Instead, most routing fees in the Lightning Network snapshot fall into a small interval, which creates several shortest paths with equal costs. Since GenPulse cannot know whether any path is optimal until it verifies all feasible paths, it will keep iterating until it finds the last shortest path. To avoid this case, we introduce an optional user-set timeout into GenPulse that forces the algorithm to return the current best solution if the timeout expires. Although the outputted path is not guaranteed to be optimal, we observe in practice that it usually is given a reasonable timeout. Besides, the user can leverage timeout expiration to increase ψ and find constrained shortest paths more efficiently.

6.5.3 MultiPulse's Performance

In our second analysis, we compare the performance of MultiPulse against several flow allocation algorithms in the literature [134, 161, 164]. The comparison consists of sending 100 payments (flows) of value $P_V \in \{10, 100, ..., 10^9\}$ in the network using different algorithms. We adopt a Lightning Network snapshot with $n = 512$ nodes and uniformly sample a set of 100 (s, t) pairs to act as the source and sink nodes of

each payment, respectively. The (s, t) pair set is the same across all runs. We repeat the experiment 30 times for each (s, t) pair to account for statistical variations. Our performance metrics are the average execution time to deliver a payment and the payment success rate expressed by:

$$PSR = \sum_{P \in \mathcal{P}} \frac{D_V}{P_V}, \tag{6.25}$$

where P is a payment from s to t with value P_V, D_V is the delivered value (the number of coins that successfully reach t), and $\mathcal{P}$ is the 100-payment set. Note that we redefined the payment success rate of Equation 5.3 such that now PSR measures the fraction of successfully delivered coins. The new definition accounts for non-atomic multipath payments in which some payment parts reach their destination while others fail. We consider a payment part k fails if: (i) any arc $(i, j) \in \pi_k$ has insufficient liquidity to forward k, where π_k is the payment part's path; or (ii) sending k through π_k yields a cost vector $\vec{c}_{\pi_k}$ that violates any side constraint.

We compare MultiPulse against three flow algorithms: Successive Shortest Paths (SSP) [164], Lightning Multipath Payments (MPP) [161], and Pickhardt Payments [134]. We provide a brief description of each algorithm below.

Successive Shortest Paths (SSP). The successive shortest paths algorithm is optimal for solving the minimum-cost flow problem [164]. It iteratively augments a flow f along the shortest paths in a residual network R_f, such that f is a maximum flow with minimum cost when no path between s and t exists in R_f. As SSP inspires MultiPulse, the algorithms are very similar, except that MultiPulse augments flows along the constrained shortest paths provided by GenPulse while SSP augments flows along the (unconstrained) shortest paths provided by some classical shortest path algorithm. In this analysis, we consider a version of SSP that finds shortest paths using Dijkstra's algorithm [144] (`spp-dijkstra`) and a version that finds shortest paths using the Bellman-Ford algorithm [145] (`spp-bf`). For both versions, the objective function the algorithm attempts to minimize is $c(f) = \sum_{(i,j) \in A} \phi_{ij} f_{ij}$, which is the default objective function of the MCF problem with $c_{ij} = \phi_{ij}$.

Lightning Multipath Payments (MPP). Lightning MPP is the default multipath payment routing algorithm for the Lightning Network. The algorithm attempts to push all the flow through the shortest path in the network and splits the payment into two parts if this attempt fails. Then, it tries to push one of the parts along the path. It does this consecutively until a part is delivered. When this happens, MPP removes the path from the network and attempts to send the remaining parts following the same process as before until all parts are delivered, or there is no path from s to t. The arc costs in MPP are a combination of routing fees, HTLC-resolution delays,

and delivery probabilities, such that $c_{ij} = \phi_{ij} + \lambda_1 \Delta_{ij} + \lambda_2 p_{ij}^{-1}$, where λ_1 and λ_2 are user-set parameters that bias the cost towards some metric. This characteristic categorizes MPP as a single mixed metric (SMM) algorithm. Besides, we can easily see that MPP is sub-optimal as it does not allocate the maximum amount of flow on each path. This behavior is intentional, as MPP privileges low execution times over flow optimality [161]. We set $\lambda_1 = 1$ and $\lambda_2 = \phi_{ij}$ for this work as these approximate the default parameters of MPP in the most popular Lightning implementation [4]. Similar to the SSP algorithm, we compare MultiPulse with a version of MPP that obtains shortest paths using Dijkstra's algorithm (`mpp-dijkstra`) and another that uses the Bellman-Ford algorithm (`mpp-bf`).

Pickhardt Payments. Pickhardt payments are a nickname for payments routed using the algorithm proposed by René Pickhardt and Stefan Richter [134]. This algorithm uses a Lagrangian relaxation method [141, 168] to move the side constraints of the CMCF problem into the objective function, effectively generating a variation of the MCF problem. The relaxation permits solving the problem with general-purpose MCF solvers [142, 143], which are presumably fast, with the downside of possibly yielding a sub-optimal solution w.r.t. the original CMCF problem. The objective function of this variation is $c(f) = \sum_{(i,j)\in A} -ln(p_{ij}) + \mu \phi_{ij} f_{ij}$, where μ is a user-set parameter that biases the cost towards routing fees. The algorithm does not consider HTLC-resolution delays. We set $\mu = 100$ in our experiments so that we optimize mostly for routing fees.

Scenario 1: Known liquidities. In the first comparison, we consider channel liquidities are known to all routing algorithms, so there is no uncertainty about arc capacities. We use this scenario both to provide insights on how each algorithm would behave if channel liquidities could be disclosed (for example, with channel probing techniques) and to analyze the maximum efficiency of each algorithm given a value of ψ.

Figure 6.7 shows the obtained results, from which we draw the following observations. First, large payments increase the execution time and reduce the payment success rate for all algorithms. This result is expected, as large payments cause algorithms to search for more paths that can carry the flow. Second, the tightness of the constraints drastically improves the performance of MultiPulse in terms of execution time to the point where it is comparable to the time performance of Lightning MPP without giving up flow optimality. Finally, MultiPulse outperforms the other algorithms in terms of payment success rates for all cases. As with GenPulse, these results confirm that MultiPulse achieves its top performance when the constraints of the CMCF problem are tight.

Scenario 2: Unknown liquidities. In our second comparison, we consider channel

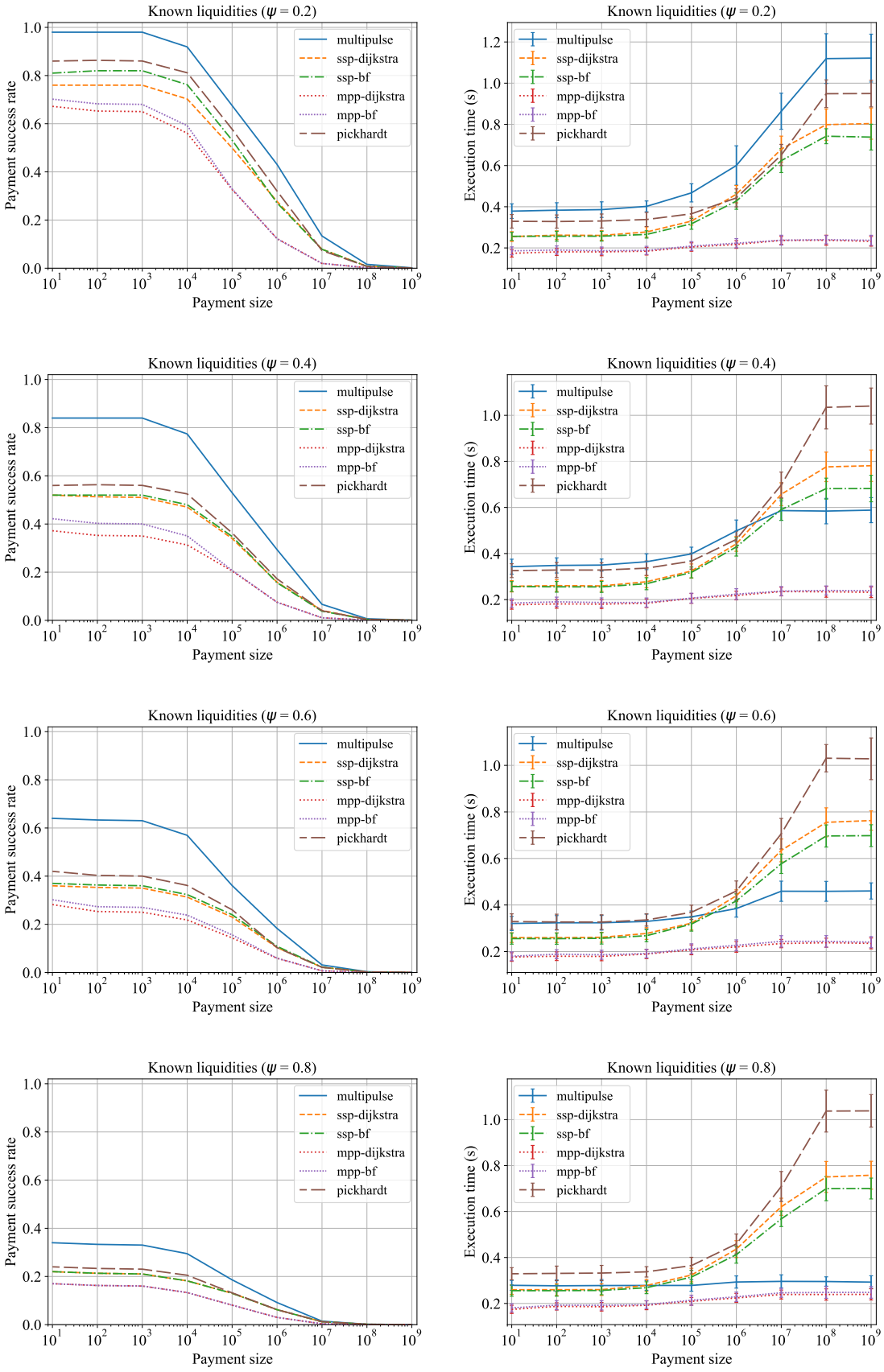

Figure 6.7: Comparison between MultiPulse and other pathfinding algorithms in the literature considering known liquidities.

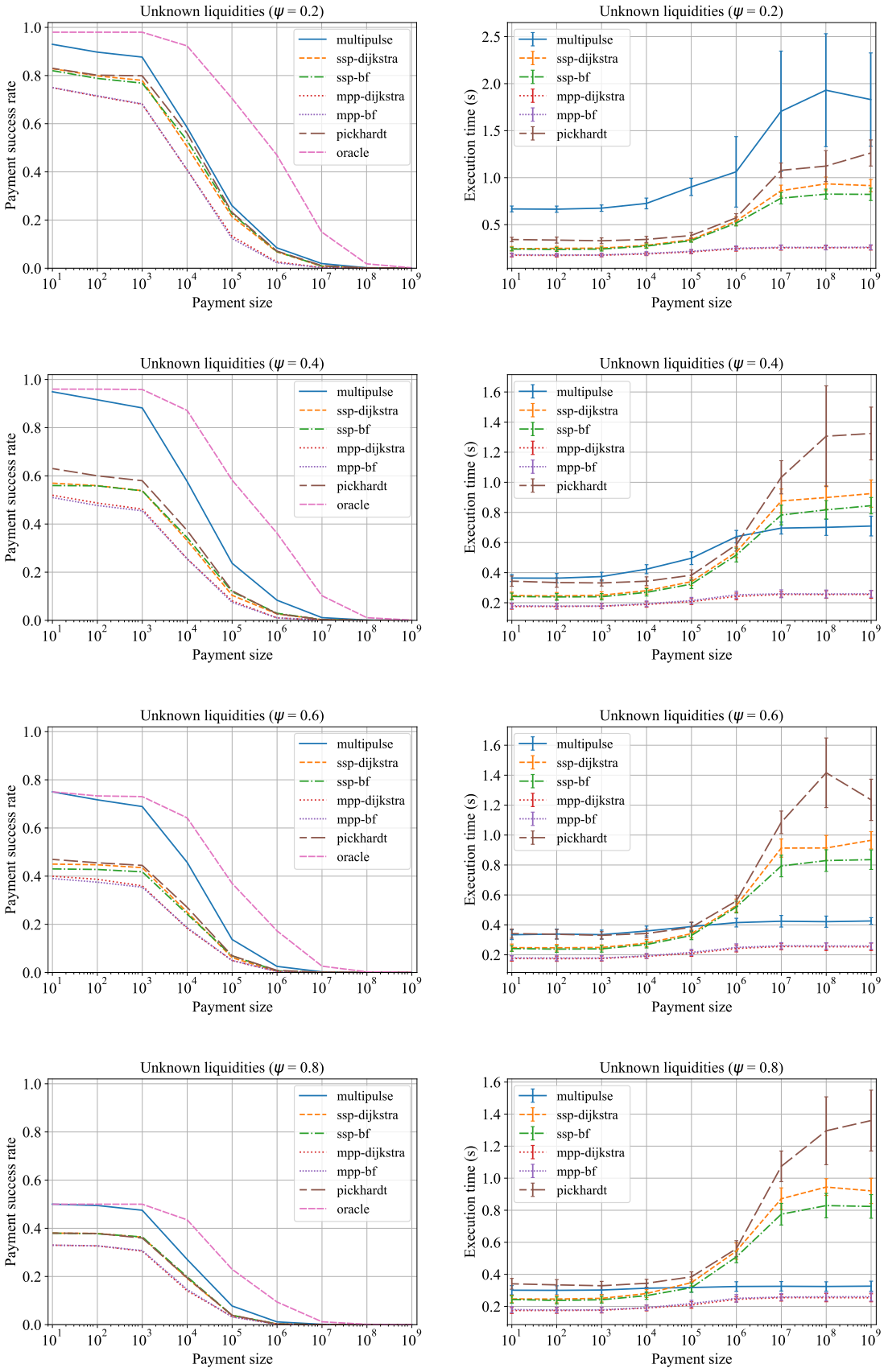

Figure 6.8: Comparison between MultiPulse and other pathfinding algorithms in the literature considering unknown liquidities.

liquidities are unknown. Unknown channel liquidities represent a realistic PCN scenario in which payment routing algorithms must estimate liquidities using an initial guess and then update the estimates according to payment results (we detailed this process in Sections 6.1.2 and 6.1.4). Thus, in each experiment, each algorithm builds an estimated liquidity network $\hat{G}_l$ such that the liquidities l_{ij} are sampled using a uniform distribution $U[0, u_{ij}]$, where u_{ij} is the capacity of the payment channel, and the corresponding opposite liquidities, l_{ji}, are set to $l_{ji} = u_{ij} - l_{ij}$. Then, the algorithm finds a feasible flow f in $\hat{G}_l$ for the first payment in the 100-payment set, attempts to send it, and updates the network according to the results of each path $\pi_k \in f$. This process repeats until the algorithm attempts all payments. Moreover, we include an optimal "oracle" algorithm that knows the actual liquidities of each arc, causing its attempts never to fail. The oracle serves as a guide to analyze how distant each algorithm's payment success rate is from the network's maximum achievable payment success rate.

Figure 6.8 shows the results. The payment success rates follow a similar pattern as in the experiment with known liquidities, with MultiPulse outperforming the other algorithms. The difference is that all algorithms, including MultiPulse, lose performance compared to the oracle. This result is a straightforward consequence of liquidity estimation errors. Moreover, we observe a subtle performance degradation concerning average execution times and standard deviations for all algorithms. This behavior is likely caused by the extra time it takes to update liquidity estimates and possibly by natural statistical variations, as the (s, t) pairs are not the same across both experiments. Like before, MultiPulse's time performance improves proportionally to the tightness of the constraints and becomes comparable to the performance of Lightning MPP when ψ is high.

6.6 Summary

In this chapter, we addressed the problem of routing payments in PCNs. We showed that the general problem of payment routing can be modeled as a minimum-cost flow (MCF) problem in a network of uncertain liquidities/arc capacities. Then, for the case in which payments come from resource-constrained devices, it can be separated into two sub-problems: (i) how to accelerate payment confirmations for light nodes, and (ii) how to route payments efficiently given a set of application-specific constraints. In the first sub-problem, we propose a secure payment scheme that confirms payments immediately and routes them later using routing services provided by capable nodes. In the second sub-problem, we show that the problem of payment routing with application-specific restrictions can be modeled as a constrained minimum-cost flow (CMCF) problem. We propose two payment routing algorithms, GenPulse and

MultiPulse, that solve the CMCF problem to find optimal constrained flows. The results indicate that both GenPulse and MultiPulse are efficient and achieve their best performances when the problem's constraints are tight.

Chapter 7

Conclusion and Future Perspectives

Current PCNs depend on capable nodes to work. Specifically, they require nodes to stay online at all times and assume nodes have enough bandwidth, storage, and computing capacities to execute the security mechanisms involved in a payment. Such assumptions are unrealistic for battery-powered devices with intermittent connectivity patterns and limited resources, which typically communicate through lossy wireless connections. Therefore, PCNs need to adapt their mechanisms to consider these devices without compromising the network's security and quality of service. The literature on this topic is relatively scarce, with only a few works proposing mechanisms that consider resource-constrained devices in their PCN model [23, 25, 105, 106].

In this thesis, we provided several contributions that, we hope, help to improve the state-of-the-art on PCNs for resource-constrained devices. Namely, we addressed the four research challenges presented in the introduction in the following ways:

- **Challenge #1 (Architecture):** We proposed a hybrid PCN architecture composed of a reliable core and peripheral unreliable light devices with limited resources. The architecture considers full nodes that perform the most computationally-intensive tasks and store blockchain copies. In contrast, light nodes communicate with them using TCP/IP connections to verify channel states on demand. The architecture reflects IoT architectures that rely on gateways and edge computing to offload tasks from resource-constrained devices.

- **Challenge #2 (Channel Security):** We proposed a simple mechanism based on minimum lock-time windows to guarantee the security of payment channels with resource-constrained devices. The mechanism considers the expected downtime and download rates of mobile broadband connections. Our main findings showed that the mechanism is most effective when the devices present high to medium availability.

- **Challenge #3 (Simulation):** We proposed PCNsim, a lightweight PCN simulator based on OMNeT++ that allows researchers to simulate payments under custom networking conditions. Our demonstrations show that PCNsim can compare routing methods and send payments over unreliable communication channels that mimic lossy wireless links between resource-constrained devices and full nodes.
- **Challenge #4 (Routing):** We proposed a payment scheme that can accelerate payment confirmations for time-sensitive applications with resource-constrained devices and proposed two pathfinding algorithms, GenPulse and MultiPulse, that can route payments optimally given application-specific constraints. The performance results of GenPulse and MultiPulse indicate that both algorithms are efficient and achieve their best performances when the problem's constraints are tight.

7.1 Open Challenges and Opportunities

Despite the recent advances, we note that PCNs are a new technology that needs to be extensively studied. Apart from the challenges explicitly addressed in this thesis, we identify several open opportunities for future researchers in the broad topic of PCNs and the specific topic of PCNs with resource-constrained devices. We provide a brief description of each opportunity below, separated into categories: i) short-term, which contains challenges that are urgent and already being addressed in the PCN community, ii) mid-term, which contains challenges which we expect will be addressed in the next couple of years, and iii)long-term, which contains challenges that are crucial to the popularization of PCNs in the future.

7.1.1 Short-term Challenges

Channel balancing. The channel liquidity distribution directly influences its capacity to forward payments and overall payment success rates. Thus, keeping channels balanced is a key concern in PCNs, whether they include light nodes or not. However, as resource-constrained devices are most likely in the network's periphery, they can hardly ever adopt common rebalancing techniques that rely on fee adjustments or circular payments [129, 130]. The usual solution is to close the channel and reopen it, which incurs transaction fees and long waiting times. Some recent proposals aim to reduce these costs in the Lightning Network [169, 170].

Node reputation systems. As we identified in Section 6.3.2, node anonymity is fundamental in PCNs and blockchains in general, but it also has the drawback of

enabling attacks that cannot be prevented without some minimal level of trust. Thus, a key challenge in PCNs is proposing an efficient node reputation system so that some nodes can be considered more trustworthy than others. Reputation systems have been proposed in a few works to prevent attacks [160, 171], but no large-scale implementation has been tested so far. For instance, a node reputation system would help improve the security of the anticipated payment confirmations we proposed for resource-constrained devices in Section 6.3.

Autopilots and network design. As PCNs rapidly grow, the research community is studying the best strategies of where to create a channel and how much to allocate to it [172–176]. Such choices are crucial both to new nodes that want to have a profitable channel and to the network as a whole, as they define how the topology will evolve [175]. Some works have used game theory to predict what network designs might emerge from such strategies and how efficient they are compared to an optimal design[1] [172–174]. However, current PCN implementations usually offer autopilots that create channels without considering such notions. Besides, this area is still incipient regarding lack large-scale studies that could help determine the actual distribution of channel-creation strategies.

7.1.2 Mid-term Challenges

Long disconnections. The minimum time-lock security mechanism against coin theft that we propose becomes inefficient for cases where the resource-constrained device disconnects for long periods (weeks or more). Besides, watchtower-based security [158], the main alternative to our approach, only works if the node can transmit a punishment transaction to the watchtower before disconnecting. Thus, we know no efficient security mechanism that can protect nodes that disconnect for long periods without warning.

Congestion control. PCNs face some challenges that are similar to traditional packet-switched networks. For example, when multiple payments traverse the same hops simultaneously, some channels may become congested or even exhausted. Concurrent payments can be especially troublesome for entry nodes connected to multiple resource-constrained devices. Since payments have a deadline to fulfill and channels might run out of capacity under heavy loads, PCNs should intelligently control the channel load to provide quality service. This topic has received little attention from the research community so far [109, 127] and thus represents a good research direction.

[1] This problem is known as "the price of anarchy" in game theory.

7.1.3 Long-term Challenges

Delay-tolerant local PCNs. As PCN nodes only ever need to interact with the blockchain for channel opening and closing, off-chain payments do not necessarily need an Internet connection to occur; it suffices that all nodes interconnect. This characteristic makes it possible to propose local PCNs that enable payments between light devices inside isolated communities, similar to delay-tolerant local blockchains [177]. These local PCNs would establish and close channels synchronously when the community connects to the Internet and perform off-chain payments anytime.

Resource-constrained core nodes. In many types of networks, e.g., mobile ad-hoc networks, vehicular ad-hoc networks, and wireless sensor networks, the resource-constrained devices act as routers for datagrams. If a PCN operates on top of these networks, the PCN's core is also composed of devices with limited resources and unreliable connections. Then, new challenges arise, such as proposing a lightweight routing protocol that considers resource consumption metrics when routing and dealing with heavy churn in the network.

7.2 Final Remarks

Given the proposals and the open challenges mentioned above, we conclude that PCNs represent an incipient area with vast research opportunities, especially concerning devices with limited resources. The results of this thesis demonstrate that this gap can and is slowly being closed so that we will see the true potential of PCNs in a few years. We hope that our studies and proposals will provide a solid foundation for future research, helping to popularize the use of cryptocurrency in the life of ordinary citizens. In future works, we intend to improve the privacy of the payment proofs in our payment scheme and to implement GenPulse and MultiPulse inside PCNsim, so we can test them under networking conditions that correspond to real environments.

References

[1] GUDGEON, L., MORENO-SANCHEZ, P., ROOS, S., et al. "SoK: Layer-two Blockchain Protocols". In: *International Conference on Financial Cryptography and Data Security*, pp. 201–226. Springer, 2020.

[2] POON, J., DRYJA, T. "The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments". 2016.

[3] RUSSELL, R., OTHERS. "BOLT #2: Peer Protocol for Channel Management". `https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md`, 2022. Last access: Mar. 6th 2023.

[4] LIGHTNING LABS. "Lightning Network Daemon". Available at `https://github.com/lightningnetwork/lnd`, 2017. Last access: Mar. 6th 2023.

[5] ELEMENTS PROJECT. "Core Lightning (CLN): A specification compliant Lightning Network implementation in C". Available at `https://github.com/lightningnetwork/lnd`, 2018. Last access: Mar. 6th 2023.

[6] ACINQ. "Eclair: A Scala implementation of the Lightning Network". Available at `https://github.com/ACINQ/eclair`, 2017. Last access: Mar. 6th 2023.

[7] ANDERSON, D., WOODRUFF, D. "Lecture #11: Network Flows III". Available at `https://www.cs.cmu.edu/~15451-f22/lectures/lec13-flow3.pdf`, 2022. Last access: Mar. 6th 2023.

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

[9] WOOD, G. "Ethereum: A Secure Decentralised Generalised Transaction Ledger". 2014. Available at <`http://bitcoinaffiliatelist.com/wp-content/uploads/ethereum.pdf`>.

[10] BLOCKCHAIN.COM. "Blockchain Charts". 2022. Available at <`https://www.blockchain.com/charts`>. Last access: Mar. 6th 2023.

[11] VISA INC. "Visa Annual Report". 2022. Available at <`https://s29.q4cdn.com/385744025/files/doc_downloads/2022/`

Visa-Inc-Fiscal-2022-Annual-Report.pdf>. Last access: Mar. 6th 2023.

[12] BUTERIN, V. "Why sharding is great: demystifying the technical properties". 2021. Available at <https://vitalik.ca/general/2021/04/07/sharding.html>. Last access: Mar. 6th 2023.

[13] HAFID, A., HAFID, A. S., SAMIH, M. "Scaling Blockchains: A Comprehensive Survey", *IEEE Access*, v. 8, pp. 125244–125262, 2020. ISSN: 2169-3536. doi:10.1109/ACCESS.2020.3007251. Conference Name: IEEE Access.

[14] SGUANCI, C., SPATAFORA, R., VERGANI, A. M. "Layer 2 Blockchain Scaling: a Survey". jul. 2021. Available at <http://arxiv.org/abs/2107.10881>. arXiv:2107.10881 [cs].

[15] GANGWAL, A., GANGAVALLI, H. R., THIRUPATHI, A. "A Survey of Layer-Two Blockchain Protocols". jul. 2022. Available at <http://arxiv.org/abs/2204.08032>. arXiv:2204.08032 [cs].

[16] YANG, D., LONG, C., XU, H., et al. "A Review on Scalability of Blockchain". In: *Proceedings of the 2020 The 2nd International Conference on Blockchain Technology*, pp. 1–6, 2020.

[17] ZHOU, Q., HUANG, H., ZHENG, Z., et al. "Solutions to Scalability of Blockchain: A Survey", *IEEE Access*, v. 8, pp. 16440–16455, 2020.

[18] SANKA, A. I., CHEUNG, R. C. C. "A systematic review of blockchain scalability: Issues, solutions, analysis and future research", *Journal of Network and Computer Applications*, v. 195, pp. 103232, dez. 2021. ISSN: 1084-8045. doi:10.1016/j.jnca.2021.103232. Available at <https://www.sciencedirect.com/science/article/pii/S1084804521002307>.

[19] CLOUDTWEAKS. "How Bitcoin Brought The Lightning Network To El Salvador". https://cloudtweaks.com/2021/07/how-bitcoin-brought-lightning-network-el-salvador/, 2021. Last access: Mar. 6th 2023.

[20] BRAINBOT LABS EST. "The Raiden Network: Fast, cheap, scalable token transfers for Ethereum". 2020. Available at <https://raiden.network/>. Available at: https://raiden.network/. Last access: Mar. 6th 2023.

[21] REED, M. G., SYVERSON, P. F., GOLDSCHLAG, D. M. "Anonymous connections and onion routing", *IEEE Journal on Selected areas in Communications*, v. 16, n. 4, pp. 482–494, 1998.

[22] ERDIN, E., MERCAN, S., AKKAYA, K. "An Evaluation of Cryptocurrency Payment Channel Networks and Their Privacy Implications". fev. 2021. Available at <http://arxiv.org/abs/2102.02659>. arXiv:2102.02659 [cs].

[23] KURT, A., AKKAYA, K., YILMAZ, S., et al. "LNGate^2: Secure Bidirectional IoT Micro-payments using Bitcoin's Lightning Network and Threshold Cryptography". jun. 2022. Available at <http://arxiv.org/abs/2206.02248>. arXiv:2206.02248 [cs].

[24] MERCAN, S., ERDIN, E., AKKAYA, K. "Improving Transaction Success Rate via Smart Gateway Selection in Cryptocurrency Payment Channel Networks". In: *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pp. 1–3, maio 2020. doi:10.1109/ICBC48266.2020.9169458.

[25] HANNON, C., JIN, D. "Bitcoin payment-channels for resource limited IoT devices". In: *IEEE COINS*, pp. 50–57, 2019.

[26] ROBERT, J., KUBLER, S., GHATPANDE, S. "Enhanced Lightning Network (off-chain)-based micropayment in IoT ecosystems", *Future Generation Computer Systems*, v. 112, pp. 283–296, 2020.

[27] CISCO. "Cisco Annual Internet Report (2018–2023) White Paper". 2020. Available at <cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf>. Last access: Mar. 6th 2023.

[28] GEISSLER, S., WAMSER, F., BAUER, W., et al. "Signaling Traffic in Internet-of-Things Mobile Networks". In: *2021 IFIP/IEEE IM*, pp. 452–458, 2021.

[29] BALTRUNAS, D., ELMOKASHFI, A., KVALBEIN, A. "Measuring the reliability of mobile broadband networks". In: *14th ACM IMC*, pp. 45–58, 2014.

[30] ELMOKASHFI, A., ZHOU, D., BALTRÜNAS, D. "Adding the next nine: An investigation of mobile broadband networks availability". In: *23rd ACM MobiCom*, pp. 88–100, 2017.

[31] OMNET++. "OMNeT++ Discrete Event Simulator". Available at https://omnetpp.org/., 2023. Last access: Mar. 6th 2023.

[32] NGUYEN, G.-T., KIM, K. "A Survey about Consensus Algorithms Used in Blockchain", *Journal of Information Processing Systems*, v. 14, n. 1,

pp. 101–128, 2018. ISSN: 1976-913X. doi:10.3745/JIPS.01.0024. Available at <https://koreascience.kr/article/JAKO201810256452304.page>. Publisher: Korea Information Processing Society.

[33] NASIR, M. H., ARSHAD, J., KHAN, M. M., et al. "Scalable blockchains — A systematic review", *Future Generation Computer Systems*, v. 126, pp. 136–162, jan. 2022. ISSN: 0167-739X. doi:10.1016/j.future.2021.07.035. Available at <https://www.sciencedirect.com/science/article/pii/S0167739X21002971>.

[34] KIM, S., KWON, Y., CHO, S. "A Survey of Scalability Solutions on Blockchain". In: *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 1204–1207, 2018.

[35] XIE, J., YU, F. R., HUANG, T., et al. "A Survey on the Scalability of Blockchain Systems", *IEEE Network*, v. 33, n. 5, pp. 166–173, 2019.

[36] KHAN, D., JUNG, L. T., HASHMANI, M. A. "Systematic Literature Review of Challenges in Blockchain Scalability", *Applied Sciences*, v. 11, n. 20, pp. 9372, jan. 2021. ISSN: 2076-3417. doi:10.3390/app11209372. Available at <https://www.mdpi.com/2076-3417/11/20/9372>. Number: 20 Publisher: Multidisciplinary Digital Publishing Institute.

[37] POPOV, S. "The Tangle", *cit. on*, p. 131, 2017. https://assets.ctfassets.net/r1dr6vzfxhev/2t4uxvsIqk0EUau6g2sw0g/45eae33637ca92f85dd9f4a3a218e1ec/iota1_4_3.pdf. Last access: Mar. 6th 2023.

[38] BAIRD, L., LUYKX, A. "The Hashgraph Protocol: Efficient Asynchronous BFT for High-Throughput Distributed Ledgers". In: *2020 International Conference on Omni-layer Intelligent Systems (COINS)*, pp. 1–7, 2020. doi:10.1109/COINS49042.2020.9191430.

[39] XIAO, Y., ZHANG, N., LOU, W., et al. "A Survey of Distributed Consensus Protocols for Blockchain Networks", *IEEE Communications Surveys & Tutorials*, v. 22, n. 2, pp. 1432–1465, 2020.

[40] REBELLO, G. A. F., CAMILO, G. F., GUIMARÃES, L. C., et al. "A security and performance analysis of proof-based consensus protocols", *Annals of Telecommunications*, pp. 1–21, 2021.

[41] CHEN, J., MICALI, S. "Algorand: A Secure and Efficient Distributed Ledger", *Theoretical Computer Science*, v. 777, pp. 155–183, 2019.

[42] EYAL, I., GENCER, A. E., SIRER, E. G., et al. "{Bitcoin-NG}: A scalable blockchain protocol". In: *13th USENIX symposium on networked systems design and implementation (NSDI 16)*, pp. 45–59, 2016.

[43] CASTRO, M., LISKOV, B. "Practical Byzantine Fault Tolerance". In: *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pp. 173–186, Berkeley, CA, USA, 1999. USENIX Association. ISBN: 1-880446-39-1.

[44] YIN, M., MALKHI, D., REITER, M. K., et al. "Hotstuff: Bft consensus with linearity and responsiveness". In: *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pp. 347–356, 2019.

[45] ANGELIS, S. D., ANIELLO, L., BALDONI, R., et al. "PBFT vs proof-of-authority: applying the CAP theorem to permissioned blockchain". In: *Italian Conference on Cyber Security (06/02/18)*, January 2018. Available at <`https://eprints.soton.ac.uk/415083/`>.

[46] THE HYPERLEDGER FOUNDATION. "Hyperledger Sawtooth". Available at `https://sawtooth.hyperledger.org/`, 2022. Last access: Mar. 6th 2023.

[47] MILLER, A., XIA, Y., CROMAN, K., et al. "The Honey Badger of BFT Protocols." In: *ACM Conference on Computer and Communications Security*, pp. 31–42. ACM, 2016. doi:10.1145/2976749.2978399.

[48] GUO, B., LU, Z., TANG, Q., et al. "Dumbo: Faster asynchronous bft protocols". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 803–818, 2020.

[49] GUO, B., LU, Y., LU, Z., et al. "Speeding dumbo: Pushing asynchronous bft closer to practice", *Cryptology ePrint Archive*, 2022.

[50] GAO, Y., LU, Y., LU, Z., et al. "Dumbo-NG: Fast asynchronous bft consensus with throughput-oblivious latency". In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1187–1201, 2022.

[51] ANDROULAKI, E., OTHERS. "Hyperledger Fabric: a distributed operating system for permissioned blockchains". In: *13th EuroSys Conference*, p. 30, 2018.

[52] CHEN, L., XU, L., SHAH, N., et al. "On security analysis of proof-of-elapsed-time (poet)". In: *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, pp. 282–297. Springer, 2017.

[53] SCHWARTZ, D., YOUNGS, N., BRITTO, A. "The Ripple protocol consensus algorithm", *Ripple Labs Inc White Paper*, 2014. https://ripple.com/files/ripple_consensus_whitepaper.pdf.

[54] LARIMER, D. "EOS.IO White Paper". 2017. Available at https://developers.eos.io/-welcome/latest/protocol/consensus_protocol. Last access: Mar. 6th 2023.

[55] AMOUSSOU-GUENOU, Y., DEL POZZO, A., POTOP-BUTUCARU, M., et al. "Dissecting tendermint". In: *International Conference on Networked Systems*, pp. 166–182. Springer, 2019.

[56] BUCHMAN, E. *Tendermint: Byzantine Fault Tolerance in the Age of Blockchains*. Tese de Doutorado, University of Guelph, 2016.

[57] YANG, F., ZHOU, W., WU, Q., et al. "Delegated Proof of Stake with Downgrade: A Secure and Efficient Blockchain Consensus Algorithm with Downgrade Mechanism", *IEEE Access*, v. 7, pp. 118541–118555, 2019.

[58] XU, J., WANG, C., JIA, X. "A Survey of Blockchain Consensus Protocols", *ACM Computing Surveys*, 2023.

[59] PAPADIS, N., TASSIULAS, L. "Blockchain-Based Payment Channel Networks: Challenges and Recent Advances", *IEEE Access*, v. 8, pp. 227596–227609, 2020. ISSN: 2169-3536. doi:10.1109/ACCESS.2020.3046020. Conference Name: IEEE Access.

[60] KHOJASTEH, H., TABATABAEI, H. "A Survey and Taxonomy of Blockchain-based Payment Channel Networks". In: *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–8, set. 2021. doi:10.1109/HPEC49654.2021.9622868. ISSN: 2643-1971.

[61] SAKAKIBARA, Y., MORISHIMA, S., NAKAMURA, K., et al. "A Hardware-based Caching System on FPGA NIC for Blockchain", *IEICE Transactions on Information and Systems*, v. 101, n. 5, pp. 1350–1360, 2018.

[62] JAVAID, H., YANG, J., SANTOSO, N., et al. "Blockchain machine: A network-attached hardware accelerator for hyperledger fabric", *arXiv preprint arXiv:2104.06968*, 2021.

[63] LIND, J., NAOR, O., EYAL, I., et al. "Teechain: A Secure Payment Network with Asynchronous Blockchain Access". In: *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, p. 63–79, New York, NY, USA, 2019. Association for Computing Machinery. ISBN: 9781450368735. doi:10.1145/3341301.3359627. Available at <https://doi.org/10.1145/3341301.3359627>.

[64] WUILLE, P., NICK, J., TOWNS, A. "Taproot: SegWit Version 1 Spending Rules". Available at https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki, 2020. Last access: Mar. 6th 2023.

[65] LOMBROZO, E., LAU, J., WUILLE, P. "BIP 141: Segregated Witness (Consensus Layer)". Available at https://www.omgwiki.org/dido/doku.php?id=dido:public:ra:xapend:xapend.b_stds:defact:bitcoin:bips:bip_0141, 2015. Last access: Mar. 6th 2023.

[66] JOURENKO, M., KURAZAMI, K., LARANGEIRA, M., et al. "SoK: A Taxonomy for Layer-2 Scalability Related Protocols for Cryptocurrencies". 2019. Available at <https://eprint.iacr.org/2019/352>. Report Number: 352.

[67] TEIXEIRA, A., NEHAB, D. "The core of cartesi", *Whitepaper, Cartesi*, 2018.

[68] BITANSKY, N., CANETTI, R., CHIESA, A., et al. "From Extractable Collision Resistance to Succinct Non-Interactive Arguments of Knowledge, and Back Again". In: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, p. 326–349, New York, NY, USA, 2012. Association for Computing Machinery. ISBN: 9781450311151. doi:10.1145/2090236.2090263. Available at <https://doi.org/10.1145/2090236.2090263>.

[69] BEN-SASSON, E., BENTOV, I., HORESH, Y., et al. "Scalable, transparent, and post-quantum secure computational integrity". Cryptology ePrint Archive, Paper 2018/046, 2018. Available at <https://eprint.iacr.org/2018/046>. https://eprint.iacr.org/2018/046.

[70] DZIEMBOWSKI, S., FAUST, S., HOSTÁKOVÁ, K. "General State Channel Networks". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, p. 949–966, New York, NY, USA, 2018. Association for Computing Machinery. ISBN: 9781450356930. doi:10.1145/3243734.3243856. Available at <https://doi.org/10.1145/3243734.3243856>.

[71] BITCOIN WIKI. "Multi-signature". Available at https://en.bitcoin.it/wiki/Multi-signature, 2021. Last access: Mar. 6th 2023.

[72] SPILMAN, J. "[Bitcoin-development] Anti DoS for tx Replacement". Available at https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002433.html, 2013.

[73] DECKER, C., WATTENHOFER, R. "A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels". In: Pelc, A., Schwarzmann, A. A. (Eds.), *Stabilization, Safety, and Security of Distributed Systems*, pp. 3–18, Cham, 2015. Springer International Publishing. ISBN: 978-3-319-21741-3.

[74] BITCOINJ.ORG. "bitcoinj". Available at https://bitcoinj.org/., 2022.

[75] BITCOINWIKI. "Script - Bitcoin Wiki". Available at https://en.bitcoin.it/wiki/Script, 2021.

[76] TOWNS, A., OTHERS. "BOLT #0: Introduction and Index". https://github.com/lightning/bolts/blob/master/00-introduction.md, 2022. Last access: Mar. 6th 2023.

[77] BITCOIN WIKI. "Hash Time Locked Contracts". Available at https://en.bitcoin.it/wiki/Hash_Time_Locked_Contracts, 2021. Last access: Mar. 6th 2023.

[78] TODD, P. "OP_CHECKLOCKTIMEVERIFY". Available at https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki, 2014. Last access: Mar. 6th 2023.

[79] ZHAO, Z., ZHOU, L., SU, C. "Systematic Research on Technology and Challenges of Lightning Network". In: *2021 IEEE Conference on Dependable and Secure Computing (DSC)*, pp. 1–8, jan. 2021. doi:10.1109/DSC49826.2021.9346275.

[80] CAMILO, G. F., REBELLO, G. A. F., SOUZA, L. A., et al. "Topological Evolution Analysis of Payment Channels in the Lightning Network". In: *IEEE Latin-American Conference on Communications (LATINCOM)*, 2022.

[81] DECKER, C. "Lightning Network Research; Topology Datasets". https://github.com/lnresearch/topology, 2021. Last access: Mar. 6th 2023.

[82] LIGHTNING NETWORK DEVELOPERS. "Lightning App Directory". Available at https://dev.lightning.community/lapps/, 2022.

[83] DINGLEDINE, R., MATHEWSON, N., SYVERSON, P. "Tor: The Second-Generation Onion Router". In: *13th USENIX Security Symposium (USENIX Security 04)*, San Diego, CA, ago. 2004. USENIX Association. Available at <https://www.usenix.org/conference/13th-usenix-security-symposium/tor-second-generation-onion-router>.

[84] MCCOY, D., BAUER, K., GRUNWALD, D., et al. "Shining light in dark places: Understanding the Tor network". In: *Privacy Enhancing Technologies: 8th International Symposium, PETS 2008 Leuven, Belgium, July 23-25, 2008 Proceedings 8*, pp. 63–76. Springer, 2008.

[85] RUSSELL, R., OTHERS. "BOLT #3: Bitcoin Transaction and Script Formats". https://github.com/lightning/bolts/blob/master/03-transactions.md#fees, 2022. Last access: Mar. 6th 2023.

[86] ZABKA, P., FÖRSTER, K.-T., SCHMID, S., et al. "Node Classification and Geographical Analysis of the Lightning Cryptocurrency Network". In: *International Conference on Distributed Computing and Networking 2021*, ICDCN '21, p. 126–135, New York, NY, USA, 2021. Association for Computing Machinery. ISBN: 9781450389334. doi:10.1145/3427796.3427837. Available at <https://doi.org/10.1145/3427796.3427837>.

[87] SERES, I. A., GULYÁS, L., NAGY, D. A., et al. "Topological analysis of bitcoin's lightning network". In: *Mathematical Research for Blockchain Economy*, pp. 1–12. Springer, 2020.

[88] LIN, J.-H., PRIMICERIO, K., SQUARTINI, T., et al. "Lightning Network: a second path towards centralisation of the Bitcoin economy", *New Journal of Physics*, v. 22, n. 8, pp. 083022, 2020.

[89] ROHRER, E., MALLIARIS, J., TSCHORSCH, F. "Discharged Payment Channels: Quantifying the Lightning Network's Resilience to Topology-Based Attacks". In: *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, pp. 347–356, 2019. doi:10.1109/EuroSPW.2019.00045.

[90] MILLER, A., BENTOV, I., KUMARESAN, R., et al. "Sprites and State Channels: Payment Networks that Go Faster than Lightning". 2017.

[91] TRINITY. "Trinity White Paper: Universal Off-chain Scaling Solution". 2018. Available at <https://www.trinity.tech/#/writepaper>. Last access: Mar. 6th 2023.

[92] INTOTHEBLOCK. "Raiden Network Statistics". 2022. Available at <https://app.intotheblock.com/coin/RDN/deep-dive?group=network&chart=all>. Last access: Mar. 6th 2023.

[93] MESSARI. "Raiden Network Market Data". 2022. Available at <https://messari.io/asset/raiden-network/metrics/all>. Last access: Mar. 6th 2023.

[94] BRAINBOT LABS EST. "Raiden Explorer". 2022. Available at <https://explorer.raiden.network/tokens>. Last access: Mar. 6th 2023.

[95] MCCORRY, P., BAKSHI, S., BENTOV, I., et al. "Pisa: Arbitration Outsourcing for State Channels". In: *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pp. 16–30, 2019.

[96] LIND, J., EYAL, I., KELBERT, F., et al. "Teechain: Scalable Blockchain Payments using Trusted Execution Environments", *arXiv preprint arXiv:1707.05454*, 2017.

[97] GREEN, M., MIERS, I. "BOLT: Anonymous Payment Channels for Decentralized Currencies". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 473–489, 2017.

[98] HOPWOOD, D., BOWE, S., HORNBY, T., et al. "Zcash protocol specification", *GitHub: San Francisco, CA, USA*, v. 4, pp. 220, 2016.

[99] HEILMAN, E., ALSHENIBR, L., BALDIMTSI, F., et al. "TumbleBit: An Untrusted Bitcoin-Compatible Anonymous Payment Hub". In: *PRoceedings of the Network and Distributed System Securty Symposium (NDSS)*, 2017.

[100] MALAVOLTA, G., MORENO-SANCHEZ, P., KATE, A., et al. "Concurrency and Privacy with Payment-Channel Networks". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[101] ERDIN, E., MERCAN, S., AKKAYA, K. "An Evaluation of Cryptocurrency Payment Channel Networks and Their Privacy Implications", *arXiv preprint arXiv:2102.02659*, 2021.

[102] CAMILO, G. F., REBELLO, G. A. F., DE SOUZA, L. A. C., et al. "A Secure Personal-Data Trading System Based on Blockchain, Trust, and Reputation". In: *2020 IEEE International Conference on Blockchain*, pp. 379–384, 2020.

[103] ROOS, S., MORENO-SANCHEZ, P., KATE, A., et al. "Settling Payments Fast and Private: Efficient Decentralized Routing for Path-Based Transactions". In: *Proceedings of the 2018 Network and Distributed System Security Symposium*. Internet Society, 2018. Available at <http://arxiv.org/abs/1709.05748>. arXiv: 1709.05748.

[104] OECD. "Mobile broadband subscriptions indicator". 2021. Available at <https://doi.org/10.1787/1277ddc6-en>. Last access: Mar. 6th 2023.

[105] KURT, A., MERCAN, S., SHLOMOVITS, O., et al. "LNGate: Powering IoT with Next Generation Lightning Micro-Payments using Threshold Cryptography". In: *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '21, pp. 117–128, New York, NY, USA, jun. 2021. Association for Computing Machinery. ISBN: 978-1-4503-8349-3. doi:10.1145/3448300.3467833. Available at <https://doi.org/10.1145/3448300.3467833>.

[106] MERCAN, S., ERDIN, E., AKKAYA, K. "Improving sustainability of cryptocurrency payment networks for IoT applications". In: *IEEE ICC Workshops*, pp. 1–6. IEEE, 2020.

[107] TOCHNER, S., ZOHAR, A., SCHMID, S. "Route Hijacking and DoS in Off-chain Networks". In: *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pp. 228–240, 2020.

[108] RILEY, G. F., HENDERSON, T. R. "The ns-3 Network Simulator". In: Wehrle, K., Güneş, M., Gross, J. (Eds.), *Modeling and Tools for Network Simulation*, Springer, pp. 15–34, Berlin, Heidelberg, 2010. ISBN: 978-3-642-12331-3. doi:10.1007/978-3-642-12331-3_2. Available at <https://doi.org/10.1007/978-3-642-12331-3_2>.

[109] SIVARAMAN, V., VENKATAKRISHNAN, S. B., RUAN, K., et al. "High Throughput Cryptocurrency Routing in Payment Channel Networks". In: *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pp. 777–796, 2020. ISBN: 978-1-939133-13-7. Available at <https://www.usenix.org/conference/nsdi20/presentation/sivaraman>.

[110] OMNET++. "INET framework". Available at https://inet.omnetpp.org/., 2023. Last access: Mar. 6th 2023.

[111] WATTS, D. J., STROGATZ, S. H. "Collective dynamics of 'small-world'networks", *nature*, v. 393, n. 6684, pp. 440–442, 1998.

[112] ALBERT, R., BARABÁSI, A.-L. "Statistical mechanics of complex networks", *Reviews of modern physics*, v. 74, n. 1, pp. 47, 2002.

[113] BARABÁSI, A.-L., BONABEAU, E. "Scale-free networks", *Scientific american*, v. 288, n. 5, pp. 60–69, 2003.

[114] PRACUCCI, M. "Linux TCP_RTO_MIN, TCP_RTO_MAX and the tcp_retries2 sysctl". Available at `https://pracucci.com/linux-tcp-rto-min-max-and-tcp-retries2.html`., 2023. Last access: Mar. 6th 2023.

[115] INET FRAMEWORK. "Packet Loss vs. Distance Using Various WiFi Bitrates". Available at `https://inet.omnetpp.org/docs/showcases/wireless/errorrate/doc/index.html`., 2023. Last access: Mar. 6th 2023.

[116] LAKSHMINARAYANAN, K., SESHAN, S., STEENKISTE, P. "Understanding 802.11 performance in heterogeneous environments". In: *Proceedings of the 2nd ACM SIGCOMM workshop on Home networks*, pp. 43–48, 2011.

[117] IEEE. "802.11g-2003 - IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Further Higher Data Rate Extension in the 2.4 GHz Band", *IEEE Std 802.11g-2003*, pp. 1–104, 2003. doi:10.1109/IEEESTD.2003.94282.

[118] HO, M.-J., WANG, J., SHELBY, K., et al. "IEEE 802.11 g OFDM WLAN throughput performance". In: *2003 IEEE 58th Vehicular Technology Conference. VTC 2003-Fall (IEEE Cat. No. 03CH37484)*, v. 4, pp. 2252–2256. IEEE, 2003.

[119] DAO, N. T., MALANEY, R. A. "Throughput performance of saturated 802.11 g networks". In: *The 2nd International Conference on Wireless Broadband and Ultra Wideband Communications (AusWireless 2007)*, pp. 31–31. IEEE, 2007.

[120] DOEFEXI, A., ARMOUR, S., LEE, B.-S., et al. "An evaluation of the performance of IEEE 802.11 a and 802.11 g wireless local area networks in a corporate office environment". In: *IEEE International Conference on Communications, 2003. ICC'03.*, v. 2, pp. 1196–1200. IEEE, 2003.

[121] CONOSCENTI, M., VETRÒ, A., DE MARTIN, J. C. "CLoTH: A Lightning Network Simulator", *SoftwareX*, v. 15, pp. 100717, jul. 2021. ISSN:

2352-7110. doi:10.1016/j.softx.2021.100717. Available at <https://www.sciencedirect.com/science/article/pii/S2352711021000613>.

[122] PIATKIVSKYI, D., NOWOSTAWSKI, M. "Split Payments in Payment Networks". In: Garcia-Alfaro, J., Herrera-Joancomartí, J., Livraga, G., et al. (Eds.), *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, Lecture Notes in Computer Science, pp. 67–75, Cham, 2018. Springer International Publishing. ISBN: 978-3-030-00305-0. doi:10.1007/978-3-030-00305-0_5.

[123] DI STASI, G., AVALLONE, S., CANONICO, R., et al. "Routing Payments on the Lightning Network". In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 1161–1170, jul. 2018. doi:10.1109/Cybermatics_2018.2018.00209.

[124] BERES, F., SERES, I. A., BENCZUR, A. A. *A Cryptoeconomic Traffic Analysis of Bitcoin's Lightning Network*. Relatório Técnico arXiv:1911.09432, arXiv, jul. 2020. Available at <http://arxiv.org/abs/1911.09432>. arXiv:1911.09432 [cs] type: article.

[125] KAPPOS, G., YOUSAF, H., PIOTROWSKA, A., et al. "An Empirical Analysis of Privacy in the Lightning Network". In: Borisov, N., Diaz, C. (Eds.), *Financial Cryptography and Data Security*, Lecture Notes in Computer Science, pp. 167–186, Berlin, Heidelberg, 2021. Springer. ISBN: 978-3-662-64322-8. doi:10.1007/978-3-662-64322-8_8.

[126] YU, R., XUE, G., KILARI, V. T., et al. "CoinExpress: A Fast Payment Routing Mechanism in Blockchain-Based Payment Channel Networks". In: *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–9, jul. 2018. doi:10.1109/ICCCN.2018.8487351. ISSN: 1095-2055.

[127] PAPADIS, N., TASSIULAS, L. "Payment Channel Networks: Single-Hop Scheduling for Throughput Maximization". In: *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, pp. 900–909, maio 2022. doi:10.1109/INFOCOM48880.2022.9796862. ISSN: 2641-9874.

[128] REBELLO, G. A. F., CAMILO, G. F., POTOP-BUTUCARU, M., et al. "PCNsim: A Flexible and Modular Simulator for Payment Channel Networks". In: *IEEE INFOCOM 2022 - IEEE Conference on Computer*

Communications Workshops (INFOCOM WKSHPS), pp. 1–2, maio 2022. doi:10.1109/INFOCOMWKSHPS54753.2022.9798003.

[129] AWATHARE, N., SURAJ, AKASH, et al. "REBAL: Channel Balancing for Payment Channel Networks". In: *2021 29th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 1–8, 2021. doi:10.1109/MASCOTS53633.2021.9614304.

[130] PICKHARDT, R., NOWOSTAWSKI, M. "Imbalance measure and proactive channel rebalancing algorithm for the Lightning Network". In: *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pp. 1–5. IEEE, 2020.

[131] AVARIKIOTI, Z., PIETRZAK, K., SALEM, I., et al. "HIDE & SEEK: Privacy-Preserving Rebalancing on Payment Channel Networks". Cryptology ePrint Archive, Paper 2021/1401, 2021. Available at <`https://eprint.iacr.org/2021/1401`>. `https://eprint.iacr.org/2021/1401`.

[132] WANG, P., XU, H., JIN, X., et al. "Flash: Efficient Dynamic Routing for Off-chain Networks". In: *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pp. 370–381, Orlando Florida, dez. 2019. ACM. ISBN: 978-1-4503-6998-5. doi:10.1145/3359989.3365411. Available at <`https://dl.acm.org/doi/10.1145/3359989.3365411`>.

[133] OSUNTOKUN, O. "[Lightning-dev] AMP: Atomic Multi-Path Payments over Lightning". Available at `https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html`, 2018. Last access: Mar. 6th 2023.

[134] PICKHARDT, R., RICHTER, S. "Optimally Reliable & Cheap Payment Flows on the Lightning Network", *arXiv:2107.05322 [cs]*, jul. 2021. Available at <`http://arxiv.org/abs/2107.05322`>. arXiv: 2107.05322.

[135] ANTONOPOULOS, A. M., OSUNTOKUN, O., PICKHARDT, R. *Mastering the Lightning Network*. O'Reilly Media, Inc., 2021.

[136] PRIHODKO, P., ZHIGULIN, S., SAHNO, M., et al. "Flare: An Approach to Routing in Lightning Network", Last access: Mar. 6th 2023, 2016.

[137] MAZUMDAR, S., RUJ, S., SINGH, R. G., et al. "HushRelay: A Privacy-Preserving, Efficient, and Scalable Routing Algorithm for Off-Chain Payments". In: *2020 IEEE International Conference*

on Blockchain and Cryptocurrency (ICBC), pp. 1–5, maio 2020. doi:10.1109/ICBC48266.2020.9169405.

[138] LIN, C., MA, N., WANG, X., et al. "Rapido: Scaling blockchain with multi-path payment channels", *Neurocomputing*, v. 406, pp. 322–332, 2020.

[139] ZHANG, Y., YANG, D., XUE, G. "CheaPay: An Optimal Algorithm for Fee Minimization in Blockchain-Based Payment Channel Networks". In: *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pp. 1–6, maio 2019. doi:10.1109/ICC.2019.8761804. ISSN: 1938-1883.

[140] ZHANG, Y., YANG, D. "RobustPay+: Robust Payment Routing With Approximation Guarantee in Blockchain-Based Payment Channel Networks", *IEEE/ACM Transactions on Networking*, v. 29, n. 4, pp. 1676–1686, ago. 2021. ISSN: 1558-2566. doi:10.1109/TNET.2021.3069725. Conference Name: IEEE/ACM Transactions on Networking.

[141] AHUJA, R. K., MAGNANTI, T. L., ORLIN, J. B. *Network flows*. Prentice-Hall, Inc., 1993.

[142] HAGBERG, A., SWART, P., S CHULT, D. *Exploring network structure, dynamics, and function using NetworkX*. Relatório técnico, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[143] BLIEK1Ú, C., BONAMI, P., LODI, A. "Solving mixed-integer quadratic programming problems with IBM-CPLEX: a progress report". In: *Proceedings of the twenty-sixth RAMP symposium*, pp. 16–17, 2014.

[144] DIJKSTRA, E. W. "A note on two problems in connexion with graphs", *Numerische mathematik*, v. 1, n. 1, pp. 269–271, 1959.

[145] BELLMAN, R. "On a routing problem", *Quarterly of Applied Mathematics*, v. 16, n. 1, pp. 87–90, 1958. ISSN: 0033-569X, 1552-4485. doi:10.1090/qam/102435. Available at <`https://www.ams.org/qam/1958-16-01/S0033-569X-1958-0102435-2/`>.

[146] LIN, C., MA, N., WANG, X., et al. "Rapido: Scaling blockchain with multi-path payment channels", *Neurocomputing*, v. 406, pp. 322–332, set. 2020. ISSN: 0925-2312. doi:10.1016/j.neucom.2019.09.114. Available at <`https://www.sciencedirect.com/science/article/pii/S0925231220305452`>.

[147] HERRERA-JOANCOMARTÍ, J., NAVARRO-ARRIBAS, G., RANCHAL-PEDROSA, A., et al. "On the Difficulty of Hiding the Balance of Lightning

Network Channels". In: *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, pp. 602–612, 2019.

[148] PICKHARDT, R., TIKHOMIROV, S., BIRYUKOV, A., et al. *Security and Privacy of Lightning Network Payments with Uncertain Channel Balances*. Relatório Técnico arXiv:2103.08576, arXiv, mar. 2021. Available at <`http://arxiv.org/abs/2103.08576`>. arXiv:2103.08576 [cs] type: article.

[149] TIKHOMIROV, S., PICKHARDT, R., BIRYUKOV, A., et al. "Probing channel balances in the lightning network", *arXiv preprint arXiv:2004.00333*, 2020.

[150] GEISSLER, S., WAMSER, F., BAUER, W., et al. "Signaling Traffic in Internet-of-Things Mobile Networks". In: *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 452–458, maio 2021. ISSN: 1573-0077.

[151] TEINTURIER, B. "Trampoline Routing". 2021. Available at <`https://github.com/lightning/bolts/pull/829`>. Last access: Mar. 6th 2023.

[152] HASBROUCK, J., SAAR, G. "Low-latency trading", *Journal of Financial Markets*, v. 16, n. 4, pp. 646–679, 2013.

[153] HAN, P., YAN, Z., DING, W., et al. "A Survey on Cross-chain Technologies", *Distributed Ledger Technologies: Research and Practice*, 2023.

[154] SIDDIQI, M. A., YU, H., JOUNG, J. "5G ultra-reliable low-latency communication implementation challenges and operational issues with IoT devices", *Electronics*, v. 8, n. 9, pp. 981, 2019.

[155] ZHU, G., WANG, Y., HUANG, K. "Broadband analog aggregation for low-latency federated edge learning", *IEEE Transactions on Wireless Communications*, v. 19, n. 1, pp. 491–506, 2019.

[156] SHE, C., SUN, C., GU, Z., et al. "A tutorial on ultrareliable and low-latency communications in 6G: Integrating domain knowledge into deep learning", *Proceedings of the IEEE*, v. 109, n. 3, pp. 204–246, 2021.

[157] XIE, T., ZHANG, J., ZHANG, Y., et al. "Libra: Succinct zero-knowledge proofs with optimal prover computation". In: *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III 39*, pp. 733–764. Springer, 2019.

[158] ION LIGHTNING NETWORK WIKI. “Watchtowers”. 2021. Available at: https://wiki.ion.radar.tech/tech/research/watchtowers. Last access: Mar. 6th 2023.

[159] REBELLO, G. A. F., POTOP-BUTUCARU, M., DE AMORIM, M. D., et al. “Securing Wireless Payment-Channel Networks With Minimum Lock Time Windows”. In: *ICC 2022 - IEEE International Conference on Communications*, pp. 2297–2302, maio 2022. doi:10.1109/ICC45855.2022.9839064. ISSN: 1938-1883.

[160] SHIKHELMAN, C., TIKHOMIROV, S. “Unjamming Lightning: A Systematic Approach”, *Cryptology ePrint Archive*, 2022.

[161] LABS, L. “Multi-Path Payments in LND: Making Channel Balances Add Up”. Available at `https://lightning.engineering/posts/2020-05-07-mpp/`, 2020. Last access: Mar. 6th 2023.

[162] LOZANO, L., MEDAGLIA, A. L. “On an exact method for the constrained shortest path problem”, *Computers & Operations Research*, v. 40, n. 1, pp. 378–384, jan. 2013. ISSN: 0305-0548. doi:10.1016/j.cor.2012.07.008. Available at <`https://www.sciencedirect.com/science/article/pii/S0305054812001530`>.

[163] PUGLIESE, L. D. P., GUERRIERO, F. “A survey of resource constrained shortest path problems: Exact solution approaches”, *Networks*, v. 62, n. 3, pp. 183–200, 2013.

[164] ENGQUIST, M. “A successive shortest path algorithm for the assignment problem”, *INFOR: Information Systems and Operational Research*, v. 20, n. 4, pp. 370–384, 1982.

[165] BRUNSCH, T., CORNELISSEN, K., MANTHEY, B., et al. “Smoothed analysis of the successive shortest path algorithm”, *SIAM Journal on Computing*, v. 44, n. 6, pp. 1798–1819, 2015.

[166] DANTZIG, G., FULKERSON, D. R. “On the max flow min cut theorem of networks”, *Linear inequalities and related systems*, v. 38, pp. 225–231, 2003.

[167] HU, P., LAU, W. C. “A survey and taxonomy of graph sampling”, *arXiv preprint arXiv:1308.5865*, 2013.

[168] JUTTNER, A., SZVIATOVSKI, B., MECS, I., et al. “Lagrange relaxation based method for the QoS routing problem”. In: *Proceedings IEEE*

INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213), v. 2, pp. 859–868 vol.2, abr. 2001. doi:10.1109/INFCOM.2001.916277. ISSN: 0743-166X.

[169] "Splicing. [Lightning-dev] Channel top-up". Available at `https://lists.linuxfoundation.org/pipermail/lightning-dev/2017-May/000696.html`, 2017. Last access: Mar. 6th 2023.

[170] LIGHTNING LABS. "Lightning Loop". 2022. Last access: Mar. 6th 2023.

[171] POPOV, S., MOOG, H., CAMARGO, D., et al. "The Coordicide". 2020. Available at <`https://files.iota.org/papers/20200120_Coordicide_WP.pdf`>. Last access: Mar. 6th 2023.

[172] AVARIKIOTI, Z., HEIMBACH, L., WANG, Y., et al. "Ride the lightning: The game theory of payment channels". In: *International Conference on Financial Cryptography and Data Security*, pp. 264–283. Springer, 2020.

[173] AVARIKIOTI, G., SCHEUNER, R., WATTENHOFER, R. "Payment Networks as Creation Games". 2019. Available at <`https://arxiv.org/abs/1908.00436`>.

[174] WANG, X., GU, H., LI, Z., et al. "Why Riding the Lightning? Equilibrium Analysis for Payment Hub Pricing". In: *ICC 2022 - IEEE International Conference on Communications*, pp. 5409–5414, 2022. doi:10.1109/ICC45855.2022.9839171.

[175] LANGE, K., ROHRER, E., TSCHORSCH, F. "On the Impact of Attachment Strategies for Payment Channel Networks", *arXiv preprint arXiv:2102.09256*, 2021.

[176] LI, P., MIYAZAKI, T., ZHOU, W. "Secure Balance Planning of Off-Blockchain Payment Channel Networks". In: *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pp. 1728–1737, 2020. doi:10.1109/INFOCOM41043.2020.9155375.

[177] HU, Y., MANZOOR, A., EKPARINYA, P., et al. "A delay-tolerant payment scheme based on the ethereum blockchain", *IEEE Access*, v. 7, pp. 33159–33172, 2019.

[178] BALAKRISHNAN, V. *Network optimization*. CRC Press, 2019.

Appendix A

Deferred Proofs

This appendix contains proofs for statements that we present throughout the text. Most proofs are adaptations of proofs we find in the literature [141, 178]. When this is the case, we indicate the corresponding original proof at the end of the adapted proof.

A.1 Flow Decomposition

Theorem 1 (Flow Decomposition). *Every non-negative flow f can be represented as a set of paths π and cycles ω that transport non-negative flows.*

Proof. The proof is algorithmic. Let s be a deficit node, i.e., a node that sends a demand d, and t be an excess node, i.e., a node that absorbs the demand. Then, at least one arc (s, i_1) carries positive flow. Start at s and visit i_1. If $i_1 \neq t$, the flow conservation constraint imposes that at least one arc (i_1, i_2) carries positive flow. Keep traversing positive-flow arcs until we reach t or a visited node i_x. If we reached t, we traversed a path $\pi = \{(s, i_1), ..., (i_n, t)\}$ that delivers some positive flow from s to t. Save the path, reassign d to $d - l_{min}$, where $l_{min} = \min(l_{ij} \forall (i, j) \in \pi)$ is the bottleneck capacity of the path, and reassign l_{ij} to $l_{ij} - l_{min} \forall (i, j) \in \pi$. If we reached a visited node, we traversed a cycle $\omega = (i_x, i_y), ..., (i_w, i_x)$ that cycles some positive flow. Save the cycle and reassign l_{ij} to $l_{ij} - l_{min} \forall (i, j) \in \omega$, where l_{min} is the bottleneck capacity of the cycle. Whenever a path or a cycle is found, restart the process at s until $d = 0$. When the algorithm stops, we have a set of paths and cycles that correspond to the original f when summed. This proof is based on Theorem 3.5 of [141]. □

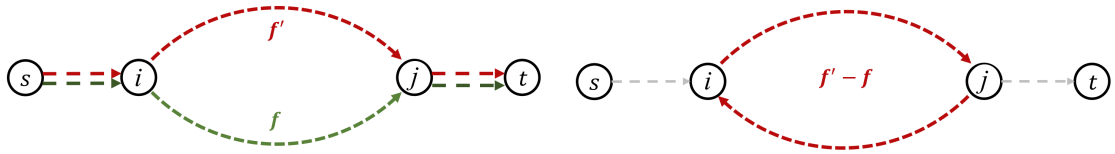

Figure A.1: An illustration of the difference between two equal-value feasible flows f' and f (adapted from [7]). The difference cancels the flow out of s and into t, creating a circulation composed of at least one cycle.

A.2 Flow Differences

Lemma 1 (Flow Differences). Let the difference $f' - f$ between two flows of equal value be a flow δ such that $\delta_{ij} = f'_{ij} - f_{ij}$ if $f'_{ij} - f_{ij} \geq 0$ and $\delta_{ji} = f_{ij} - f'_{ij}$ if $f'_{ij} - f_{ij} < 0$. The difference δ yields a circulation composed of at least one augmenting cycle ω in the residual graph R_f.

Proof. From the definition of δ, there are two possible outcomes for any arc $(i, j) \in R_f$ that has positive flow w.r.t. f': have its flow reduced to $f'_{ij} - f_{ij}$ if $f'_{ij} \geq f_{ij}$ or be reversed into an arc (j, i) with $f_{ji} = f_{ij} - f'_{ij}$ if $f'_{ij} < f_{ij}$. As f' and f are feasible flows, they satisfy the flow conservation property, and hence their difference δ also satisfies the flow conservation property for any node j in the network. For any j, this implies that if some incoming arc (i, j) is reduced or inverted, then some outgoing arc (j, k) will surely suffer a similar process to conserve the flow. Moreover, because f' and f are equally valued by assumption, the net flow $f' - f$ going out of the source s and into the sink t must be zero. Consequently, any non-zero flows going out of a node j other than s and t must eventually come back into j, creating a circulation of augmenting cycles. We illustrate this process graphically in Figure A.1. This proof is based on Theorem 2 of [7].

□

A.3 Negative Cycle Optimality

Theorem 2 (Negative Cycle Optimality). *Let ω be a negative-cost cycle with net cost $c(\omega) = \sum_{(i,j)\in\omega} c_{ij} < 0$. A flow f^* is a minimum-cost flow if and only if no ω exists in the residual graph R_f.*

Proof. Suppose that f is a feasible flow and that R_f contains a negative-cost cycle ω. Then, f is not optimal as we can obtain a flow f^* with lower cost by augmenting f along ω. Therefore, if f^* is an optimal cost flow, R_{f^*} cannot contain negative-cost cycles. Now, suppose that f^* is a feasible flow such that R_{f^*} contains no negative-cost cycles and let a flow $f' \neq f^*$ be the optimal cost flow. The flow difference lemma

(Lemma 1) shows that we can represent the difference $f^* - f'$ as a set of augmenting cycles in R_{f^*} and that the sum of the costs of such cycles is $c(f') - c(f^*)$. As there are no negative-cost cycles in R_{f^*}, then $c(f') - c(f^*) \geq 0$, or $c(f') \geq c(f^*)$. Moreover, because f' is optimal by assumption, $c(f') \leq c(f^*)$. Thus, $c(f') = c(f^*)$, proving that f^* is also optimal. This proof is based on Theorem 9.1 of [141]. □

A.4 Optimality of MultiPulse

Proposition 1 (Optimality of MultiPulse). *The MultiPulse algorithm provides an optimal solution f^* to the constrained minimum-cost flow problem.*

Proof. Let R_f be the resulting residual network after a flow f is pushed through a transportation network G and π_σ^* be the shortest path in R_f w.r.t. the main metric σ in some iteration. We aim to prove that no negative cost cycles are created in R_f during MultiPulse's execution, as this is a criterion of optimality in transportation networks (Theorem 2). When the current flow has zero value, R_f contains no negative cost cycles by definition since we assume this of G. Now, suppose that in some iteration of MultiPulse, the algorithm creates a negative cost cycle ω after pushing an amount through π_σ^*. This implies that some arc $(i, j) \in \pi_\sigma^*$ closed the cycle ω when reversed and that such reversed arc (j, i) yields a negative cost c_{ji} that compensates for all the other (positive) costs of in the cycle, i.e., $c_{ji} + \sum_{(u,v)\in\omega} c_{uv} < 0$. However, we can see that if such an arc (i, j) exists, then the shortest path would not traverse it since traversing the other arcs in the cycle would yield a lower cost $\sum_{(u,v)\in\omega} c_{uv} < c_{ij}$ in the first place. This contradicts the assumption that π_σ^* is the shortest path, proving that ω could never be created if π_σ^* is indeed optimal. Furthermore, because the paths π_σ^* are constrained shortest paths provided by GenPulse, it is also guaranteed that side constraints are never violated. Consequently, the solution f^* given by the last iteration of MultiPulse is an optimal solution to the constrained minimum-cost flow problem. □

Appendix B

List of Publications

The following works were published during the elaboration of this thesis:

- **Rebello, G. A. F.**, Camilo, G. F., Guimarães, L. C. B., Souza, L. A. C., Duarte, O. C. M. B., "On the Security and Performance of Proof-based Consensus Protocols", in 4th Conference on Cloud and Internet of Things (CIoT 2020), Niterói, Brazil, October 2020.

- **Rebello, G. A. F.**, Camilo, G. F., Guimarães, L. C. B., Souza, L. A. C., Duarte, O. C. M. B., "Segurança e Desempenho de Protocolos de Consenso Baseados em Prova para Corrente de Blocos", in XX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg 2020), Petrópolis, Brazil, October 2020.

- **Rebello, G. A. F.**, Camilo, G. F., Guimarães, L. C. B., de Souza, L. A. C., Thomaz, G. A., Duarte, O. C. M. B. – "A Security and Performance Analysis of Proof-based Consensus Protocols", in Annals of Telecommunications, no. 7, pp. 517-537, 2021.

- **Rebello, G. A. F.**, Camilo, G. F., Guimarães, L. C. B., Souza, L. A. C., Duarte, O. C. M. B. - "Security and Performance Analysis of Quorum-based Blockchain Consensus Protocols", in 6th Cyber Security in Networking Conference (CSNet'22) - Rio de Janeiro, Brazil, October 2022.

- **Rebello, G. A. F.**, Camilo, G. F., Souza, L. A. C., Potop-Butucaru, M., Amorim, M. D., Campista, M. E. M., Costa, L. H. M. K. - "A Survey on Layer-Two Protocols for Blockchains", submitted to IEEE Communications Surveys & Tutorials in April 2023.

- **Rebello, G. A. F.**, Potop-Butucaru, M., Amorim, M. D., Duarte, O. C. M. B. – "Protegendo Redes de Canais de Pagamento Sem Fio com Janelas de Tempo

de Bloqueio Mínimas", XXI Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg 2021), Belém, Brazil, October 2021. Honorable mention.

- **Rebello, G. A. F.**, Potop-Butucaru, M., Amorim, M. D., Duarte, O. C. M. B. - "Securing Wireless Payment-Channel Networks With Minimum Lock Time Windows", IEEE International Conference on Communications (ICC 2022), Seoul, South Korea, May 2022.

- **Rebello, G. A. F.**, Potop-Butucaru, M., de Amorim, M. D., Duarte, O. C. M. B. "Sécurisation des réseaux de canaux de paiement sans fil avec des fenêtres de temps de verrouillage réduites". In CORES 2022–7ème Rencontres Francophones sur la Conception de Protocoles, l'Évaluation de Performance et l'Expérimentation des Réseaux de Communication.

- **Rebello, G. A. F.**, Camilo, G. F., Potop-Butucaru, M., Campista, M. E. M., Amorim, M. D., Costa, L. H. M. K. - "PCNsim: A Flexible and Modular Simulator for Payment Channel Networks", IEEE International Conference on Computer Communications Workshops (INFOCOM WKSHPS), Virtual Conference, May 2022.

- Camilo, G. F., **Rebello, G. A. F.**, de Souza, L. A. C., Campista, M. E. M., Costa, L. H. M. K. – "Posicionamento Lucrativo de Nós e Criação de Rotas de Baixo Custo na Rede Relâmpago", in XLI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2023), Brasília, DF, Brazil, May 2023. Honorable mention.

- de Souza, L. A. C., **Rebello, G. A. F.**, Camilo, G. F., Campista, M. E. M., Costa, L. H. M. K. – "GITI-CB: Gestão de Identidade com Troca de Informações entre Correntes de Blocos", in VI Workshop Blockchain: Teoria, Tecnologia e Aplicações (Wblockchain 2023), Brasília, DF, Brazil, May 2023.

- de Souza, L. A. C., Camilo, G. F., **Rebello, G. A. F.**, Sammarco M., Campista, M. E. M., Costa, L. H. M. K. – "ATHENA-FL: Evitando a Heterogeneidade Estatística através do Um-contra-Todos no Aprendizado Federado", in VII Workshop de Computação Urbana (CoUrb), Brasília, DF, Brazil, May 2023.

- Camilo, G. F., **Rebello, G. A. F.**, de Souza, L. A. C., Thomaz, G. A., Potop-Butucaru, M., Amorim, M. D., Campista, M. E. M., Costa, L. M. K. – "Redes de Canais de Pagamento: Provendo Escalabilidade para Pagamentos em Criptomoedas", in Minicursos do XL Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2022), Fortaleza, Brazil, May 2022.

- Camilo, G. F., **Rebello, G. A. F.**, de Souza, L. A. C., Potop-Butucaru, M., Amorim, M. D., Campista, M. E. M., Costa, L. H. M. K. – "Análise da Evolução Topológica da Rede Lightning de Canais de Pagamento", in XXII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg 2022), Santa Maria, Brazil, September 2022.

- Camilo, G. F., **Rebello, G. A. F.**, de Souza, L. A. C., Campista, M. E. M., Costa, L. H. M. K. – "Posicionamento Lucrativo de Nós e Criação de Rotas de Baixo Custo na Rede Relâmpago", in XLI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2023), Brasília, DF, Brazil, May 2023. Honorable mention.

- de Souza, L. A. C., **Rebello, G. A. F.**, Camilo, G. F., Campista, M. E. M., Costa, L. H. M. K. – "GITI-CB: Gestão de Identidade com Troca de Informações entre Correntes de Blocos", in VI Workshop Blockchain: Teoria, Tecnologia e Aplicações (Wblockchain 2023), Brasília, DF, Brazil, May 2023.

- de Souza, L. A. C., Camilo, G. F., **Rebello, G. A. F.**, Sammarco M., Campista, M. E. M., Costa, L. H. M. K. – "ATHENA-FL: Evitando a Heterogeneidade Estatística através do Um-contra-Todos no Aprendizado Federado", in VII Workshop de Computação Urbana (CoUrb), Brasília, DF, Brazil, May 2023.

- Camilo, G. F., **Rebello, G. A. F.**, de Souza, L. A. C., Thomaz, G. A., Potop-Butucaru, M., Amorim, M. D., Campista, M. E. M., Costa, L. M. K. – "Redes de Canais de Pagamento: Provendo Escalabilidade para Pagamentos em Criptomoedas", in Minicursos do XL Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2022), Fortaleza, Brazil, May 2022.

- Camilo, G. F., **Rebello, G. A. F.**, de Souza, L. A. C., Potop-Butucaru, M., Amorim, M. D., Campista, M. E. M., Costa, L. H. M. K. – "Análise da Evolução Topológica da Rede Lightning de Canais de Pagamento", in XXII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg 2022), Santa Maria, Brazil, September 2022.

- **Rebello, G. A. F.**, Hu, Y., Thilakarathna, K., Batista, G. E. A. P. A., Seneviratne, A., and Duarte, O. C. M. B. – "Melhorando a Acurácia da Detecção de Lavagem de Dinheiro na Rede Bitcoin", in XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2020), Rio de Janeiro, Brazil, December 2020.

- Camilo, G. F., **Rebello, G. A. F.**, Souza, L. A. C., Duarte, O. C. M. B., "A Secure Personal-Data Trading System Based on Blockchain, Trust, and

Reputation", in 3rd IEEE International Conference on Blockchain (Blockchain-2020), Rhodes Island, Greece, November 2020.

- Camilo, G. F., **Rebello, G. A. F.**, Souza, L. A. C., Duarte, O. C. M. B. – "AutAvailChain: Disponibilização Segura, Controlada e Automática de Dados IoT usando Corrente de Blocos", in III Workshop em Blockchain: Teoria, Tecnologias e Aplicações (WBlockchain SBRC 2020), Rio de Janeiro, Brazil, December 2020. Honorable mention.

- Souza, L. A. C., **Rebello, G. A. F.**, Camilo, G. F., Guimarães, L. C. B., Duarte, O. C. M. B., "DFedForest: Decentralized Federated Forest", in 3rd IEEE International Conference on Blockchain (Blockchain-2020), Rhodes Island, Greece, November 2020.

- Camilo, G. F., **Rebello, G. A. F.**, Souza, L. A. C., Duarte, O. C. M. B., "Um Sistema Seguro de Comercialização de Dados Pessoais Sensíveis baseado em Reputação, Confiança e Corrente de Blocos", in XX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg 2020), Petrópolis, Brazil, October 2020.